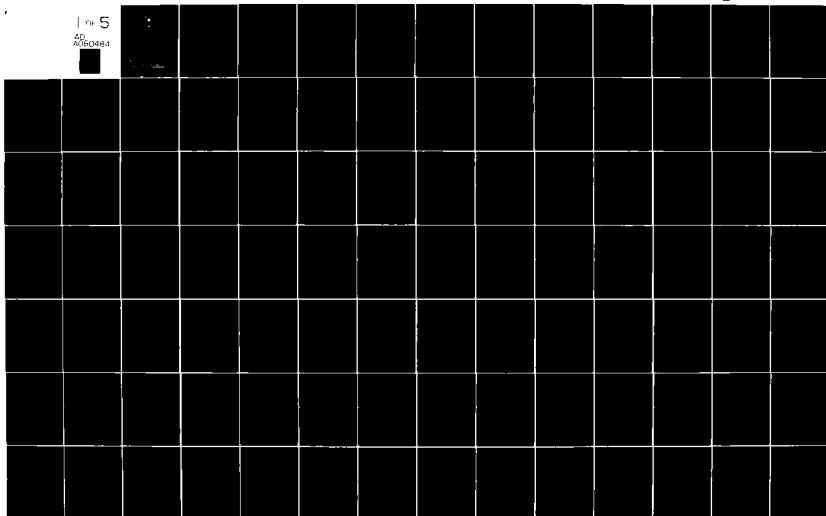


AD-A080 484 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL--ETC F/6 17/2  
AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II.(U)  
DEC 79 J A WHITTENTON  
UNCLASSIFIED AFIT/SCS/EE/79-15 NL

1 of 5

AD-A080484



AD A 080484



0  
LEVEL II  
R

AN ANALYSIS OF A ROUTING ALGORITHM  
FOR AUTODIN II

THESIS

APIT/GCS/EE/79-15 John A. Whittento  
Captain USA

DDC FILE COPY

DISTRIBUTION STATEMENT A

Approved for public release  
Distribution Unlimited

DDC  
RECEIVED  
FEB 11 1980  
A

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY (ATC)

**AIR FORCE INSTITUTE OF TECHNOLOGY**

THIS DOCUMENT IS BEST QUALITY PRACTITIONER  
THE COPY FURNISHED TO DDC CONTAINED A  
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT  
REPRODUCE CORRECTLY.  
Wright-Patterson Air Force

80 2 5 246

AFIT/GCS/EE/79-15

(6)  
AN ANALYSIS OF A ROUTING ALGORITHM  
FOR AUTODIN II

(7) THESIS

(14) AFIT/GCS/EE/79-15

(1) John A. Whittenton  
Captain USAF

1980

DISTRIBUTION STATEMENT A

Approved for public release  
Distribution Unlimited

012 222

## **DISCLAIMER NOTICE**

**THIS DOCUMENT IS BEST QUALITY  
PRACTICABLE. THE COPY FURNISHED  
TO DDC CONTAINED A SIGNIFICANT  
NUMBER OF PAGES WHICH DO NOT  
REPRODUCE LEGIBLY.**



AFIT/GCS/EE/79-15

AN ANALYSIS OF A ROUTING ALGORITHM  
FOR AUTODIN II

THESIS

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology  
Air University (ATC)  
in Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science

by

John A. Whittenton  
Captain USAF

Graduate Computer Systems

December 1979

Accession For	<input checked="checked" type="checkbox"/>
REF ID: A1	
EDITION	
UNCLASSIFIED	
Justified	
By	

A 23  
CP

Approved for public release; distribution unlimited.

## Preface

The development of a routing algorithm for the new Automatic Digital Network II (AUTODIN II) has been a topic of major concern for two years. The Defense Communications Agency (DCA) contracted out to Western Union Telegraph Company the design and development of the AUTODIN II network configuration and systems engineering. As authorized through Western Union by DCA, Systems Control Incorporated (SCI) was contracted to develop a routing algorithm based on the Advanced Research Projects Agency Network (ARPANET) routing algorithm but tailored specifically to the AUTODIN II network.

In essence, this research project is a computer simulation of the SCI network routing algorithm to determine its characteristics and limitations under varying degrees of network configuration and system stress conditions.

I would like to express my thanks to Capt Wade L. Neilson for his help in gathering the technical data on the algorithm. In addition, I want to express my sincere appreciation to Dr. Tom Hartrum and Capt Larry Kizer whose technical expertise proved invaluable. A very special thanks goes to my thesis advisor, Major Allen A. Ross, for his constructive advice, technical assistance and a sincere interest in the project from its conception to completion. My deepest appreciation goes to my loving wife, Lorraine, whose moral support and sympathetic understanding induced me to continue throughout the course of the project.

John A. Whittenton

## Contents

	<u>Page</u>
Preface .....	ii
List of Figures .....	vi
List of Tables .....	viii
Abstract .....	ix
An Analysis of a Routing Algorithm for AUTODIN II	
I. Introduction .....	1
Background .....	1
Purpose/Problem .....	2
Scope .....	3
Organization .....	3
II. The AUTODIN II Network .....	5
Basic Computer Networks .....	5
Network Description .....	6
Network Architecture .....	7
Topology/Connectivity .....	7
Node Decomposition .....	9
III. An Overview of Routing Algorithms .....	14
Non-Adaptive Control .....	14
Centralized Adaptive Control .....	16
Isolated Adaptive Control .....	16
Distributed Adaptive Control .....	17
IV. Hierarchical Routing Algorithm Description ....	19
Algorithm Database .....	22
Modules of the Algorithm .....	32
PAKROUTE .....	36
MESGEN .....	36
UPDATE .....	40
RTABLE .....	42
TTABLE .....	42
MINHOP .....	42
Background Functions .....	45
Per Packet Functions .....	54
V. GPSS Simulation Model .....	57
Language Description .....	57
Program Description .....	62

## Contents Cont'd

	<u>Page</u>
VI. Initial Baseline Considerations .....	75
Packet Interarrival Times and .....	
Length Determinations .....	76
Priority, Length and Destination .....	
Assignments .....	85
Packet Processing and Transmission .....	90
Running the Baseline Model .....	97
Baseline Results .....	102
VII. Algorithm Description .....	111
Fortran Subroutines for Modules .....	111
Simulation Program Modifications .....	115
GPSS HELP Blocks .....	115
Algorithm Activation .....	116
Procedure for Running the Modified .....	
Program .....	124
Trunk Loss Simulation .....	124
Link Loss Simulation .....	130
Node Loss Simulation .....	134
VIII. Simulation Results and Analysis .....	146
Algorithm Adaptability .....	146
Selected Results Explained .....	147
Fully Operational Algorithm Run vs .....	
Baseline .....	148
Simulated Topology Degradations .....	150
Sample Trunk Loss .....	151
Sample Link Loss .....	152
Sample Node Loss .....	153
Summary Tables .....	154
IX. Conclusions/Recommendations .....	165
Conclusions .....	165
Recommendations .....	166
Bibliography .....	168
Appendix A: Baseline Simulation Code for the .....	
Andrews Node .....	170
Appendix B: Sample of Hierarchical Algorithm .....	
Coded in Fortran .....	179

## Contents Cont'd

	<u>Page</u>
Appendix C: Albany's Definitions of Subroutine ..... Table, Array, Constant, and Variable ..... Names .....	192
Appendix D: Full Network Simulation Program .....	195
Appendix E: Listing of SAVEVALUES .....	257
Appendix F: Graphical and Tabular Results from the ... Fully Operational Network .....	266
Appendix G: Graphical and Tabular Delay Results ..... from the Trunk Loss Runs .....	272
Appendix H: Graphical and Tabular Delay Results ..... from the Link Loss Runs .....	303
Appendix I: Graphical and Tabular Delay Results ..... from the Nodal Loss Runs .....	352
Vita .....	371

## List of Figures

Figure		Page
1	The AUTODIN II Network .....	8
2	AUTODIN II Inter-Node Connectivity .....	11
3	Packet Paths at a Node .....	12
4	Initial LD Table for the Andrews Node .....	25
5	P and NC Arrays .....	26
6	HOP Arrays for Andrews .....	28
7	LIST Array for Andrews .....	29
8	D Tables for Andrews .....	31
9	Overview of Routing Module .....	33
10	PAKROUTE .....	37
11	MESGEN .....	38
12	UPDATE .....	41
13	RTABLE .....	43
14	CONNECTIVITY .....	46
15	INITIALIZE_D .....	47
16	COMPUTE_D .....	49
17	CONN_CHANGE .....	51
18	Logical Order of Background Tasks .....	52
19	Schematic View of Per Packet Tasks .....	55
20	Network Layout of AUTODIN II .....	63
21	GPSS Block Diagram for Andrews .....	64
22	System Delay (All Packets) .....	104
23	System Delay (Non-Priority - Short Packets) ..	105
24	System Delay (Non-Priority - Long Packets) ...	106
25	AUTODIN II Packet Delay Specifications .....	107

List of Figures Cont'd

Figure		Page
26	Code Necessary to Drop Trunk in ..... Albany - Ft. Detrick Link .....	126
27	Albany's CONN_CHANGE Subroutine .....	128
28	Code for Loss of Gentile - Ft. Detrick Link ..	131
29	CONN_CHANGE Subroutine for Gentile .....	132
30	Example of Code for Loss and Return of a Node.	136
31	PAKROUTE for Albany .....	140
32	PAKROUTE for Ft. Detrick .....	143

### List of Tables

Table		Page
I.	Table of Variables .....	34
II.	Subscriber Input by Node .....	79
III.	Normalized Subscriber Input .....	80
IV.	Packet Input Rates and Interarrival Times ...	83
V.	Node -to-Node Packet Exchange .....	87
VI.	Node-to-Node Packet Exchange Ratios .....	88
VII.	Network Base Routing Matrix .....	92
VIII.	Example Output of System Delay Table RTIME ..	95
IX.	System Delay (Priority - Short Packets) .....	108
X.	System Delay (Priority - Long Packets) .....	109
XI.	Disturbance Period Summary .....	156
XII.	Excessive Mean Delay Summary for .....	
	Non-Priority - Short Packets .....	158
XIII.	Excessive Mean Delay Summary for .....	
	Non-Priority - Long Packets .....	160
XIV.	Algorithm Behavioral Ability .....	163



### Abstract

In computer communications systems, a network routing algorithm is the methodology behind which a message is directed (routed) from a source computer site, through the network of linked computers, to a destination computer site.

The simulation of such an algorithm is of extreme importance in determining the efficiency of the algorithm, its reaction response to network topology changes, and its system response as the network capacity of messages reaches the saturation point.

The algorithm to be simulated is hierarchical in nature, that is, with multiple processors at each computer site (node), the best route to the succeeding node is selected first, then a processor having a physical communications line (trunk) in the best route is selected. The connectivity of the network (how the nodes are connected to one another) and the amount of congestion at each node are vital parameters used by the algorithm in determining the best route.

The simulation language selected was General Purpose Simulation System (GPSS). It is an event oriented language readily adaptable to network queueing models. Entities called transactions are generated to simulate message segments (packets) which are then processed as a normal packet would be in a communications processor. Time delays are incurred by the transaction due to processing of the packet, wait time in queues and transmission delays (the time it takes to transmit the packet from one node to another).

The actual algorithm is written in FORTRAN. GPSS instruction blocks are used to jump from the GPSS code to the FORTRAN subroutines where the actual path is calculated that the packet will take to the next node.

A baseline program with static routing, that is, fixed routing that does not change, was first developed to establish a control program.

Four levels of input packet generation were used to vary the network loading from 60% to 90% of network trunk saturation. In addition to varying the traffic volume through the network, selected transmission trunks and links, and nodes were removed from the network to simulate varying topology changes.

AN ANALYSIS OF A ROUTING ALGORITHM  
FOR AUTODIN II

I Introduction

Background

A new computerized defense communications network, Automated Digital Network II (AUTODIN II), is currently being designed by the Defense Communications Agency (DCA). The new system is a result of increased user requirements on the current AUTODIN I network, additional independent military Automated Data Processing (ADP) systems requiring computer communications support, and projected computer-oriented military information systems requesting entry into the World Wide Military Command and Control System (WWMCCS) Standard ADP System (Ref 1:1-1). In 1972, DCA began conducting a comprehensive study to identify current and future ADP requirements. The study concluded that by 1976 there would be approximately 250 computers involved in on-line communications functions with approximately 8,000 input/output terminals (Ref 1:1-1). Projections into the mid 1980's indicated that within the Department of Defense (DoD) there will be approximately 2,500 computers involved in communications with some 20,000 input/output terminals (Ref 1:1-1). The approach to AUTODIN II is start small (8 computer switching centers), meet the known validated requirements, and grow/evolve as requirements increase and

the needs for service change.

In June 1975 the Director, Telecommunications and Command and Control Systems (DTACCS), Office of the Secretary of Defense, gave initial program approval for AUTODIN II's development. The Request for Proposals was released to industry on 25 November 1975, with proposals due by 7 April 1976; final program approval was obtained in November 1976 with subsequent contract award to the Western Union Telegraph Company and its major subcontractors, Ford Aerospace and Communications Corporation and Computer Services Corporation (Ref 1:1-2). The Government issued the order to proceed with design and implementation of an initial four computer network on 14 January 1977 (Ref 1:1-2).

AUTODIN II is a digital, common-user, distributed communications computer network employing a message transmittal methodology called message packet-switching. A message is segmented into finite length segments called packets. Each packet contains all the essential information necessary for routing processes in addition to the actual data to be transmitted. The procedures or processes which direct (route) a message or packet from a source computer site through the network of linked computers, to a destination computer site are called routing algorithms.

#### Purpose/Problem

A routing algorithm has been developed by Systems

Control Incorporated similar to the one used by the Advanced Research Projects Agency Network (ARPANET) but with added refinements to tune it to the AUTODIN II network. An algorithm developed for use in a highly sophisticated and complex network should be able to handle variations (spiked increases) in normal message volume with packets accumulating only minor additional delay time. It should also be able to detect any topological change or congestion occurring in the network and reroute the packet accordingly, with minimum additional delay in packet delivery. The purpose of this thesis is to determine the behavior characteristics of the routing algorithm under various simulated conditions involving the introduction of spikes in message volume and changes in network topology.

### Scope

The content of this report is limited exclusively to the description, incorporation of the algorithm into the simulation model, and execution of the AUTODIN II routing algorithm. Detailed technical descriptions of other network system components are beyond the scope of this paper.

A network model is developed with a static (non-changing, deterministic) routing table as a control model. The algorithm will then be incorporated into the network simulation model.

### Organization

This report contains nine chapters. Chapter II begins

with a brief explanation of computer networks in general, then proceeds with a description of the AUTODIN II network architecture. Network topology, packet-switch center (node) decomposition, and network connectivity are also discussed. Chapter III describes the types of routing algorithms used in various computer networks. Chapter IV describes the developed hierarchical routing algorithm in detail. The six modules composing the algorithm and their interrelationships are discussed. The two modes of the algorithm, "Per Packet Functions" and "Background Functions" are also covered.

Chapter V discusses the simulation language, General Purpose Simulation System (GPSS), and the network simulation model. In Chapter VI, the baseline simulation model is developed with discussions on the static routing table, packet generation, source node traffic routing matrix and other essential baseline considerations.

Chapter VII deals with how the algorithm is incorporated into the network simulation model. Traffic volume changes and topological changes are also explained.

Chapter VIII discusses the results of the simulation with emphasis on packet delay analysis. In Chapter IX conclusions on the algorithm's adaptability to network changes and congestion, and any abnormal results are stated. Recommendations on possible changes or modifications to the algorithm or its documentation and any advisable network reconfigurations are also included in Chapter IX.

## II The AUTODIN II Network

### Basic Computer Networks

In general, a computer network is formed from the tying or linking together of two or more computers with their associated communications interfaces. The computers that process and direct messages or packets along to their destinations are called switching computers or nodes. The locations of the nodes can be geographically dispersed (that is, a site on the East Coast and a site on the West Coast), or locally connected (that is, intercity). Connecting the nodes together are physical data communications links called trunks. The trunks are made up of leased land lines, microwave links, satellite channels, or any combination of these.

Host computers and/or terminals are usually connected to the nodes. The network services the host computers and terminals by directing or routing the data messages to and from other host computers and terminals either connected to the same node or located at some other distant node. The primary purpose of the network is to permit access by a user, or by a process acting on behalf of the user, to resources of the host computers (Ref 2,48). The primary function of the network is to provide the medium through which the user's data messages are input from his equipment, properly routed to their destination node, and output to the destination's equipment. The network should be invisible to the users, providing only a virtual connection between

the user's input equipment and the Host computer the user is querying.

#### Network Description

The proposed AUTODIN II network is the result of several comprehensive communications computer requirements surveys (Ref 1:1-1). There are numerous existing and near-future computer systems employing various techniques for common-user communications between various military organizations. The surveys consolidated the different organizational requirements, near future and tentative computer systems requirements for communications computer support. The end result was the AUTODIN II network (Ref 3).

AUTODIN II is designed to provide economical and reliable data communications service both for interactive time-sharing and transaction-oriented systems requiring rapid response between terminals and computer-to-computer data transfers requiring high transmission capacity (Ref 3). From the source or initial node, each packet is dynamically routed along one of several alternate paths, in some cases passing through one or more tandem nodes before arriving at the destination node. Unlike AUTODIN I which requires a message to be received in its entirety at a node before transmitting the message further, AUTODIN II provides the capability of routing simultaneously packets (containing part of the original message) independently of other packets of the same message thus propagating the message through the



network faster. If a message is made up of multiple packets, each packet can be routed independently of the other packets. Thus several packets can be propagated through the network at the same time, not necessarily using the same route to get to the destination node. If there are two separate paths to get to the destination, half the packets can be transmitted on one path, the other half being transmitted on the other path. In effect, the packets parallel each other in arriving at their destination node. A packet incurs a delay of only a very small fraction of a second at each node, so that the total delay, from the time a packet fully enters the initial node from the user (subscriber) until it begins to leave the destination node, shall be less than a second (Ref 3).

#### Network Architecture

Topology/Connectivity. The network is currently made up of eight packet-switching nodes (PSNs) located at Albany, GA; Andrews AFB, MD; Ft. Detrick, MD; Gentile AFS, OH; Hancock Field, NY; McClellan AFB, CA; Norton AFB, CA; and Tinian AFB, OK as Figure 1 depicts. The numbers in the figure represent how many trunks are contained in each link. These also are the sites of the current AUTODIN I network. The decision was made to colocate the two networks for ease of maintainability, cutover procedures, and obvious budget considerations.

The connectivity of the network is an important factor

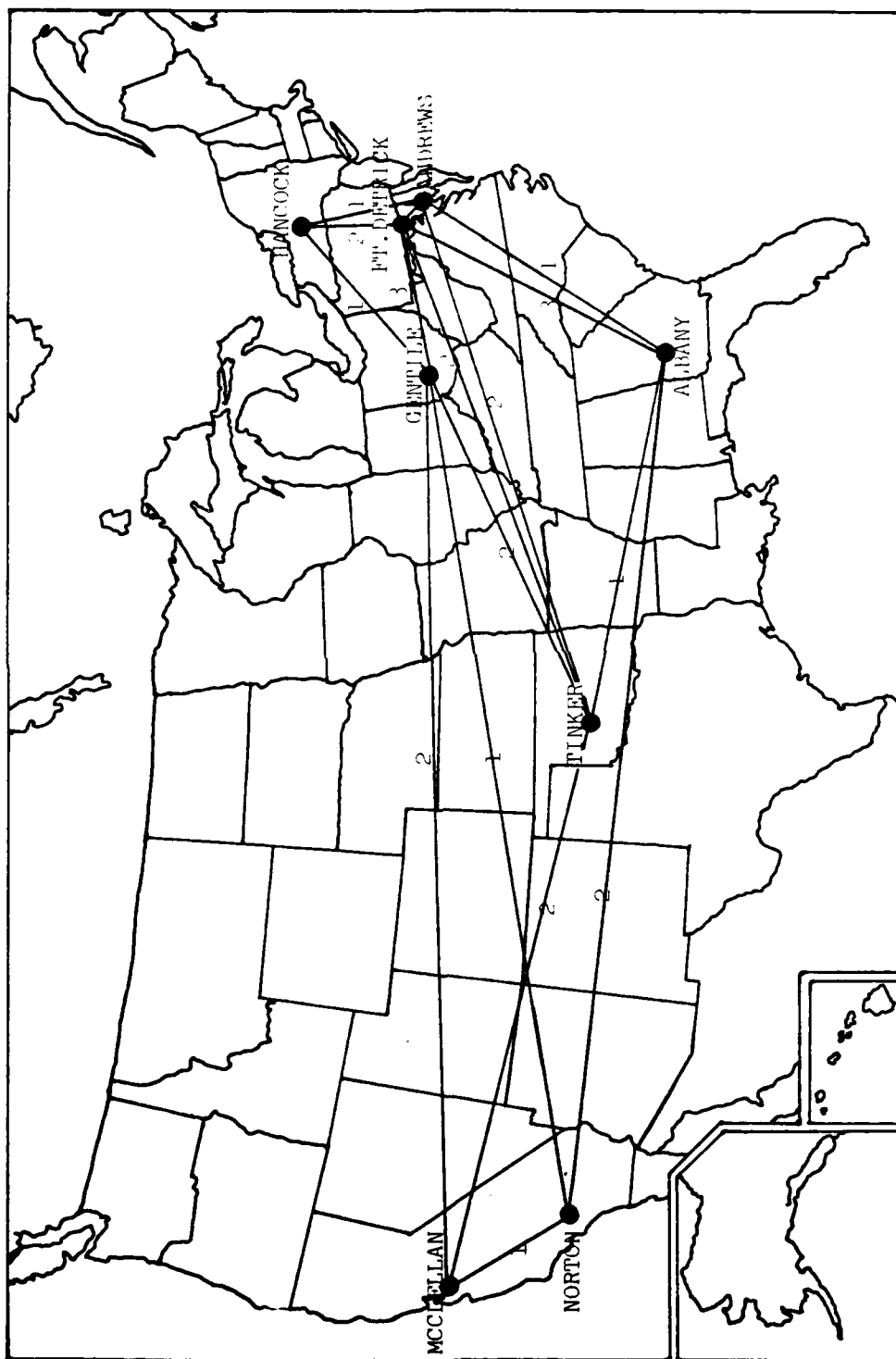


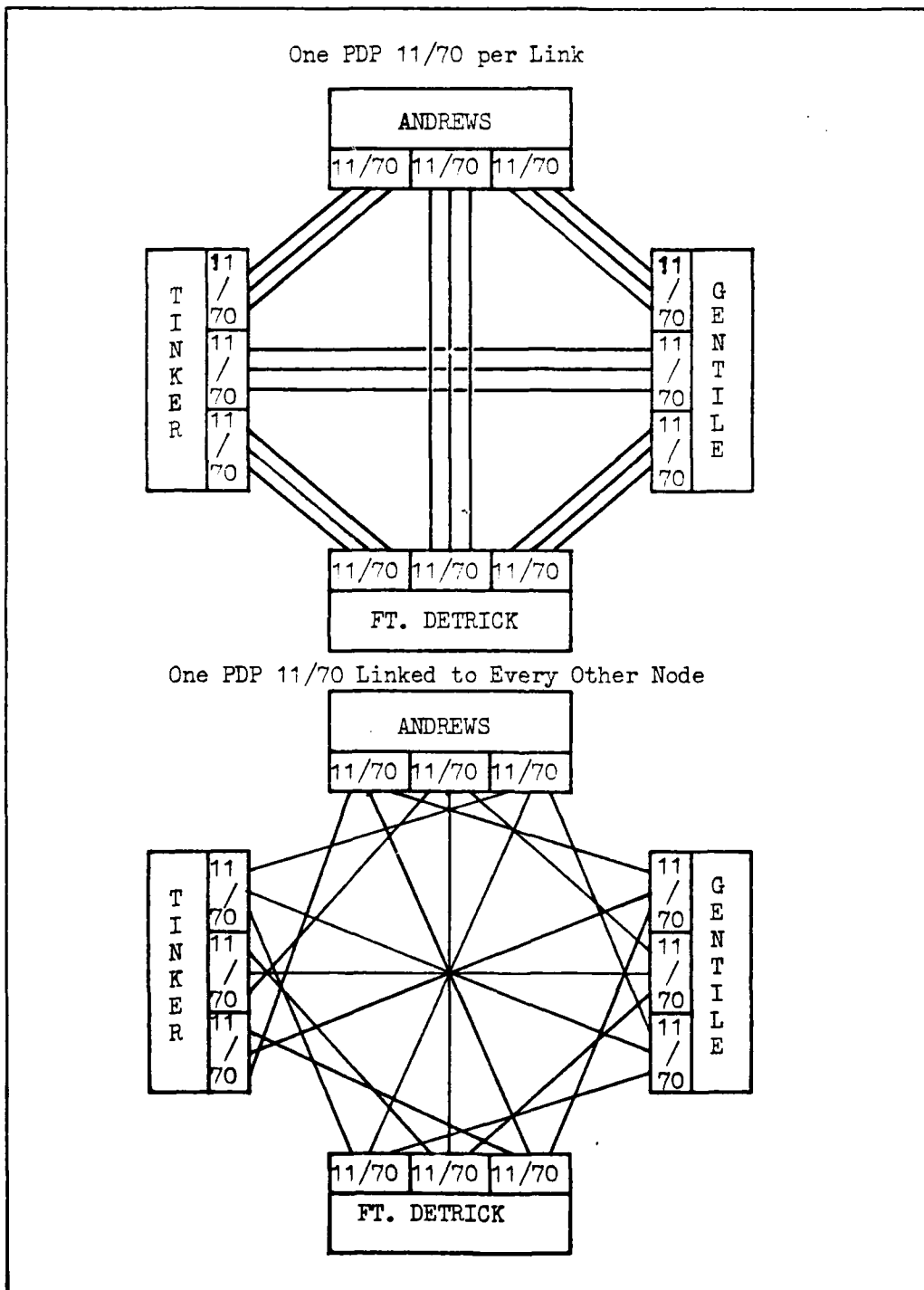
Fig 1. The AUTODIN II Network

in the network. If each node in the network has three other nodes connected to it, then the connectivity of the network is defined to be 3.0. Unlike the ARPANET with connectivity 2.4 (2.4 nodes connect to each node), there are an average of four other nodes connecting each node in the AUTODIN II network representing a connectivity of 4.0. Packet wise, this means that there are fewer nodes, and therefore fewer links, for a packet to traverse in order to arrive at its destination node. The average number of hops (link traversals) a packet takes in the AUTODIN II network is .84, whereas in ARPANET it is 6 (Ref 3). The result of high network connectivity is less time delay from packet entry into the source node to delivery at the destination node. High network connectivity, as in the AUTODIN II network, means a packet requires fewer hops to get to its destination. With each hop, a packet incurs delay by having to be input processed by the receiving node and also by having to be retransmitted to the next node. The fewer hops a packet must make, the smaller the delay is at arriving at its destination.

Node Decomposition. Each packet-switching node (PSN) contains from one to three PDP 11/70 processors (Switch Control Modules (SCMs)), depending on the particular PSN's average traffic volume. Interconnecting the nodes are from three to five logical communications links. Each logical link contains from one to three physical transmission lines (trunks), again depending on the traffic volume of the site.

If all trunks of each PSN-PSN link were terminated on the same SCM, the routing of packets within the node could be done on a deterministic basis. However, in order to insure that the load is evenly distributed over SCMs at the node, to minimize dual processing required because of node connectivity, and to provide higher access to the available transmission bandwidth, the trunks for each link are distributed across SCMs at each node (Ref 4:25). This is shown in Figure 2 with the bottom figure representing the selected method of tying SCMs together. NOTE: This figure is for example only. Each node does not have three SCMs and the nodes are not totally connected in the real network. The example is shown to convey the idea of even trunk distribution.

The connectivity of the Host computers and terminals into a node is called the access area. The term backbone pertains to the routing of packets solely between nodes. There are four basic paths for packets to take in the overall network system: (1) access area-to-access area, (2) access area-to-backbone, (3) backbone-to-backbone, and (4) backbone-to-access area. These paths are shown in Figure 3. In Figure 3a, packets enter the node from the access area, processed, and routed locally back to the access area. Figure 3b shows packets entering the node from the access area, being processed and routed to another node via the network backbone. Figure 3c depicts the path for packets entering the node from another node via the network



**Fig 2. AUTODIN II Inter-Node Connectivity**

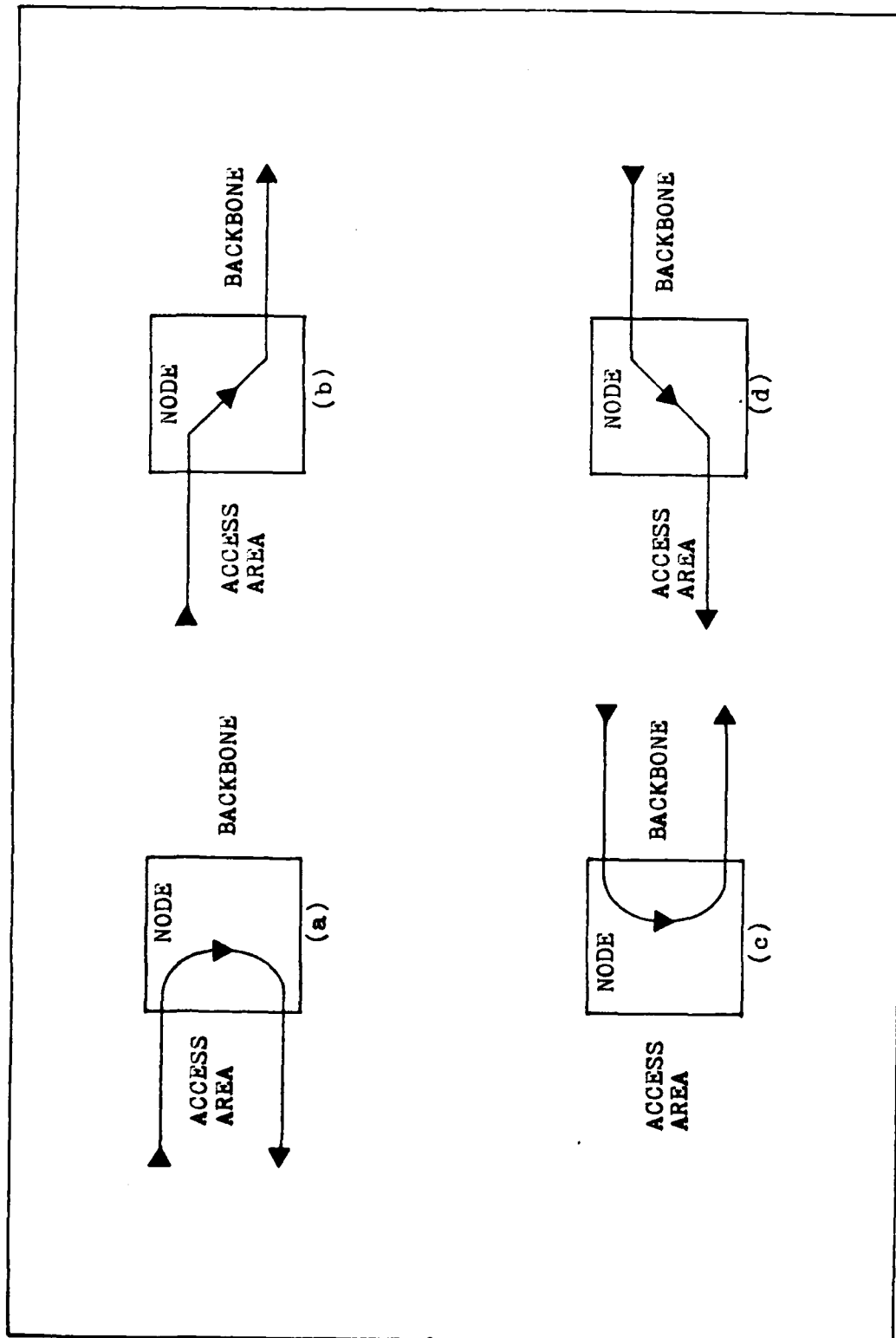


Fig 3. Packet Paths at a Node

backbone, processed and routed to another node via the backbone. In Figure 3d, packets enter the node via the backbone from another node, processed and routed to a local destination via the access area.

### III An Overview of Routing Algorithms

The basic philosophy behind a message or packet routing function is the establishment of a continuous path, usually incorporating several transmission lines (trunks) of a network, between any pair of source and destination nodes, along which messages are transmitted (Ref 5:213). The implementation of the route chosen consists of setting up, at each node along the path, a routing table that directs messages with a particular destination address to the appropriate outgoing trunk at the node. The table and associated table look-up procedures are incorporated in the memory and software of the computer at the node in question (Ref 5:213).

A fundamental element underlying all routing algorithms is the choice of the control ideology under which the algorithm is executed. The four most widely known and accepted ideologies are non-adaptive, centralized, isolated, and distributed (Ref 6).

#### Non-Adaptive Control

Non-adaptive or fixed algorithms are deterministic (static) in nature, that is, they do not adapt or change to take into consideration vacillations in traffic congestion or topological changes at the node. Rather, they are designed to provide satisfactory performance, on the average, over a range of traffic intensities (Ref 5:214). A message's path through the network is uniquely predetermined from



its source node to its destination node. Routing tables, predefined at each node, contain the path a message takes for any unique source-destination node pair. The routing algorithm generally used for this type of control ideology is called a shortest path type of algorithm. For each unique source-destination node pair, one path is generated. The path is broken into segments, each node in the path being responsible for routing the message over the particular segment it is in charge of. SITA, a cooperative company of international air carriers, has established a data communications network that employs this type of routing algorithm (Ref 5:28).

For the reader's information, AUTODIN I maintains static (fixed) routing tables. The entries to the routing tables are input one at a time, by hand-cut paper tape. A routing command/change is entered at the operator's console which reads the routing changes on the paper tape. An on-line utility program makes the correct changes to the routing tables. The routing table consists of primary, secondary, and in some cases tertiary routes. If the primary route for a message is disabled, an ALTRROUTE (alternate route) command is entered from the operator's console which automatically establishes the secondary route as the primary route. When the disabling condition is corrected, a command is entered at the console which reestablishes the primary route.

### Centralized Adaptive Control

Centralized adaptive algorithms utilize some kind of central supervisory program which dictates the routing decisions to the individual nodes in response to network changes (Ref 6). TYMNET, a computer-communications network developed in 1970, executes this class of routing algorithm. One example, the least-cost algorithm, as employed by TYMNET, is executed every time a user connects into the network and selects the route determined to be the least-cost for the user, then transmits this selected route to the nodes involved ahead of time (Ref 5:27). The routing algorithm, as contained in the supervisory program, finds the path of current least-cost, summed over the costs of each link on the path, from source computer to destination computer (Ref 5:27).

### Isolated Adaptive Control

Isolated adaptive algorithms operate independently of other nodes in the network with each node making exclusive use of local data to adapt to changing conditions (Ref 6). The data usually contain parameters on source node, destination node, local trunk queue lengths, local topology, and message priority. From the data, a delay table and a routing table are formed. The delay time estimate entries in the delay table are the number of messages waiting in the queue on the particular outgoing link. The delay table is indexed by the destination node

and outgoing links. If a link connecting two nodes contain more than one trunk, the message with the highest priority is always routed on the trunk with the shortest queue. For non-priority messages, either trunk can be selected if the difference in the queue lengths is not greater than a set weighted bias. The bias can be adjusted to fit the needs of the node and the level of traffic volume. From the delay table, the link with the smallest delay is inserted in the routing table. The routing table is indexed by the destination node and in the case of a multi-trunk link, the trunks with the shortest and next shortest queues.

#### Distributed Adaptive Control

By far the most widely used and the subject of current state-of-the-art research, is the distributed adaptive routing algorithm. It utilizes internode cooperation and exchange of information to arrive at routing decisions (Ref 6). Several techniques are combined to make up this type of algorithm. First of all, the main feature of the algorithm involves the exchange of delay information between neighboring nodes. Whether the delay information is determined on estimated delay times to all the network nodes or just the delay to neighboring nodes, it is this information which the routing algorithm uses to update local routing tables at the node. Second, added to these delay estimates are local delay values based on the number of messages or packets on queue awaiting transmission on local trunks.

Third, these combined delays are compared against the current corresponding entries in the routing table and are substituted for the current entries if the new delays are less than the current delay. And fourth, the process of local decision routing becomes inherent in the algorithm by virtue of the fact that local queue lengths take an active part in the delay calculations. The ARPANET employs a version of this type of routing algorithm with delay status messages exchanged with neighboring nodes on the order of every 625 milliseconds (Ref 5:48).

The routing algorithm proposed for the AUTODIN II network is very similar to the ARPANET algorithm but contains four major differences. First, status information is exchanged between all network nodes rather than with neighboring nodes. Second, the information exchanged reflects local link congestion and/or local link connectivity status, that is, whether or not the link is up or down rather than the estimated delay to each of the other nodes. Third, the exchange rate of status packets is either periodical (every 400 milliseconds), event driven when link congestion occurs, or when a change in link status has been determined rather than set at every 625 milliseconds. And fourth, changes in the status of links or nodes which cause the generation of status packets produce a network topological table resident at each node making that node cognizant of the status of each of the other nodes in the network, whereas the ARPANET does not maintain a topological table at all.

#### IV Hierarchical Routing Algorithm Description

Chapter IV describes in detail the different tables generated and used by the routing algorithm. It also contains a detailed explanation of each module making up the algorithm. After reading Chapter IV, the reader should have a good grasp of how the algorithm builds and uses its various tables in order to arrive at the best trunk over which to route a packet to its appropriate destination. The reader should also have a feel of how each module fits into the overall operation of the algorithm.

The hierarchical routing algorithm is a distributed adaptive algorithm utilizing both status information exchanged between nodes in the network and local output queue lengths to determine the best path over which to route a packet. It derives its name from the two levels of routing procedures executed to determine the final trunk over which a packet is transmitted.

The unique quality requiring two levels of decision making is due to the presence of multiple (one to three) communications processors called Switch Control Modules (SCMs), located within each node of the network. At present, AUTODIN II is the only packet-switching network to employ multiple processors to handle the routing of packets through a network. Conditioned on predicted traffic volumes per site, one to three SCMs (PDP 11/70s) handle the routing decisions. Neighboring nodes connected by one

logical link but that link is made up of one to three physical transmission lines (trunks).

The first level of decision making selects the route or logical link over which the packet will be transmitted. The second level selects which SCM to use which, in effect, selects the individual trunk.

When a packet is received at a node, some process or set of processes must be executed to determine the next route the packet has to take to reach its final destination. The hierarchical routing algorithm is made up of such a set of processes to determine the best route for the packet to follow. The algorithm contains processes that maintain and update various tables in the algorithm's database used to calculate the best route. Route congestion or failure along with nodal congestion or failure are detected by the algorithm and compensated for by a recalculation of its database tables. Network awareness of the degradation is facilitated by the cognizant nodes formulating and transmitting status messages to the other nodes in the network. Upon reception of the status messages, a node uses the information in the status messages to recalculate its tables.

Due to the configuration of the trunk connections to the SCM(s), different routing results are obtainable. In one instance, a node contains one SCM and the selected link resulting from the execution of the algorithm contains multiple trunks. The trunk that has the shortest queue is

selected. In the case where the node contains multiple SCMs and the selected link is made up of only one trunk, the packet could incur additional delay required to switch the packet to the SCM that is connected to the correct trunk. That additional delay is not incurred if the SCM processing the packet is also the SCM to which that trunk is connected. In the situation where the node contains two SCMs and the selected link contains two trunks one connected to each of the SCMs, a bias factor is involved in the trunk selection because of the possibility of having to switch SCMs. If the packet is routed to a trunk on another SCM within the same node, information specifying which trunk queue is to be used goes along with it. Thus packets arriving at an SCM from another SCM within the same node can be queued directly for the correct trunks (Ref 7:9).

The operation of the routing algorithm can be summarized as follows (Ref 8:1-2):

- (1) Each node maintains a copy of the network topology.
- (2) Each node monitors its output trunk group queue, and compares them with a predetermined congestion threshold value.
- (3) Whenever the immediate connectivity of a node alters, or the congestion status of any of its trunk group alters, or on a periodical basis, the node transmits a connectivity/congestion message to all other nodes.

- (4) On receipt of a connectivity/congestion message, each node updates its local copy of the network topology and updates its table of minimum-hop paths to all destinations.
- (5) Tandem nodes (nodes between the source and destination nodes) route packets by choosing only among the minimum-hop paths.
- (6) Source nodes route packets by choosing among the minimum-hop paths and the minimum-hop + 1 paths.
- (7) When a tandem node or source node has two or more paths to a destination, it chooses a specific output link by minimizing the combination of local queue delays plus bias representing the delay due to any reported congestion on the paths.

#### Algorithm Database

Each node operates with a copy of the algorithm. There are several tables and arrays calculated and maintained at each node by the algorithm. The tables are the Link Distance (LD) table and the Routing Distance (D) table. The LD table represents the nodal topology of the network. The AUTODIN II network is not a totally connected network, that is, each node is not connected to every other node. The LD table maintains which nodes are connected to which nodes along with the identification of congested links between nodes.



The D table contains, for each link a node maintains, the number of link traversals (hops) necessary to reach a given destination node.

The arrays the algorithm maintains and uses are the Connectivity Pointer (P) array, Connectivity Vector (NC) array, HOP array, LIST array, SROUTE array, and the ROUTE array. The P and NC arrays are interrelated arrays used in conjunction with other arrays to build the D table. The HOP array contains one entry per node in the network. Each entry is the number of hops required to get to the respective node from the local node. The LIST array also contains one entry per node in the network. Each entry is the respective node's ID number. The SROUTE and ROUTE arrays are used to select the link, given the packet destination node, over which to route the packet. SROUTE is used if the node is the originating node of the packet and ROUTE is used if the node is an intermediate node.

The Link Distance (LD) table is an  $N \times N$  matrix where  $N$  is the number of nodes in the network, in this case eight. Nodes are numbered from one to eight in alphabetical order from Albany to Tinker. Source nodes correspond to row headings and destination nodes correspond to column headings. An entry in the table is a tri-state quantity reflecting either connectivity with no congestion, connectivity with congestion, or no connectivity. For example, let node  $i$  and node  $j$  be two nodes in the network. If there is a link between them and it is not congested,

the LD (i,j) entry is 1.0. If there is a link between the two nodes but it is congested beyond a set congestion threshold, the corresponding LD (i,j) entry is a 1.01. If the two nodes are not connected the LD (i,j) value is a pseudo infinity value (1000) which, in practice, is a very large number to preclude any packet from being queued to a non-existent link. Figure 4 is representative of the initial LD for the Andrews node.

The two arrays P and NC are interrelated arrays. The P array (connectivity pointer array), is an  $N+1 \times 1$  vector whose entries point to entries in the NC array. The NC array (connectivity vector) is an  $NL \times 1$  vector where NL is the number of one way links in the network, in this case thirty-two. The first NLN1 entries, where NLN1 is the number of links Albany maintains, are the numerical representations of the nodes connected to Albany. The next NLN2 entries, where NLN2 is the number of links Andrews maintains, are the numerical representations of the nodes connected to Andrews and so on for the rest of the nodes and their respective entries in the NC array. The entries of the P array point to the entry of the NC array that corresponds to the first link of a particular node. Figure 5 illustrates the contents of both the P and the NC arrays.

The HOP array is an  $N \times 1$  array whose entries correspond to the minimum number of hops required for a packet to reach a particular node. The source node's entry is a zero.

		D E S T I N A T I O N    N O D E (j)							
		ALBANY	ANDREWS	FT. DETRICK	GENTLE	HANCOCK	MCLELLAN	NORTON	TINKER
j:	k	1	2	3	4	5	6	7	8
ALBANY	1	1000	1	1	1000	1000	1000	1	1
ANDREWS	2	1	1000	1	1000	1	1000	1000	1
FT. DETRICK	3	1	1	1000	1	1	1000	1000	1
GENTLE	4	1000	1000	1	1000	1	1	1	1
HANCOCK	5	1000	1	1	1	1000	1000	1000	1000
MCLELLAN	6	1	1000	1000	1	1000	1000	1	1
NORTON	7	1	1000	1000	1	1000	1	1000	1000
TINKER	8	1	1	1	1	1000	1	1000	1000

Fig 4. Initial LD Table for the Andrews Node

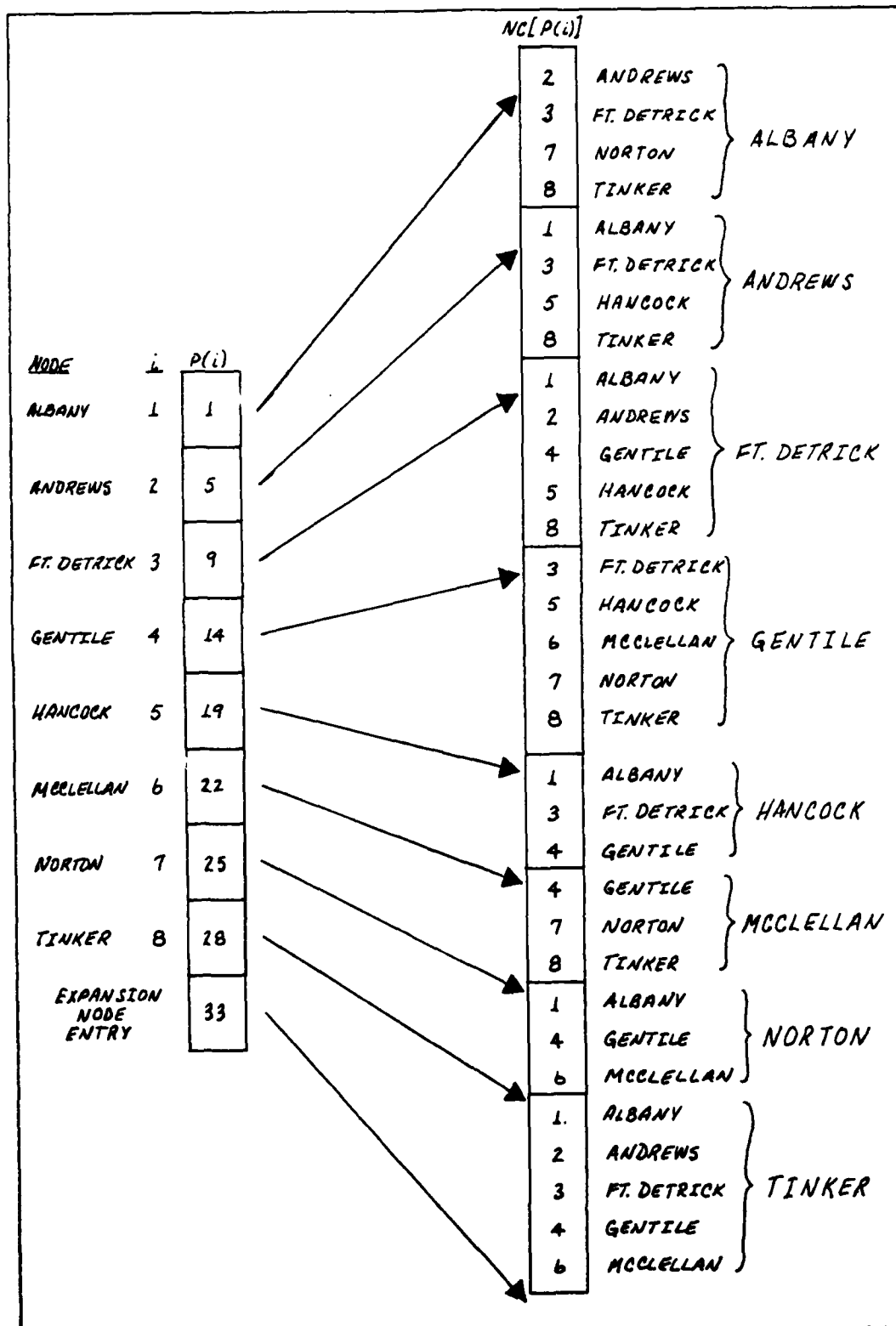


Fig 5. P and NC Arrays

Initially, the entries, except the source node entry, are either 1s or pseudo infinities, the 1s corresponding to directly connected nodes and the pseudo infinities corresponding to non-connecting nodes. Figure 6 shows both the initial HOP array and the revised version after initialization for the Andrews node.

The LIST array is an  $NL \times 1$  array. Each node is given a unique integer identification number from one to eight. The ID number is assigned in numerical order to the nodes arranged in alphabetical order, that is, Albany-1, Andrews-2, ..., Tinker-8. The first NLN entries, where NLN is the number of links a particular node maintains, are the numerical representations (ID numbers) of the nodes connected to the particular node executing the algorithm. They are basically taken from the NC array. The other  $N - NLN - 1$  entries are the numerical representations of the nodes not connected. There is no entry for the source node. If, during the operation of the algorithm, a connected node's link drops out, that node's numerical value is negated, removed from the set of connected node's values, inserted among the set of non-connected nodes. that link is flagged as being disconnected. Re. Figure 7 for the initial LIST array and the revised version after initialization for the Andrews node.

The LD table and the P, NC, HOP, and LIST arrays are combined to form the most important table the algorithm maintains, the D table (routing distance table). It is an

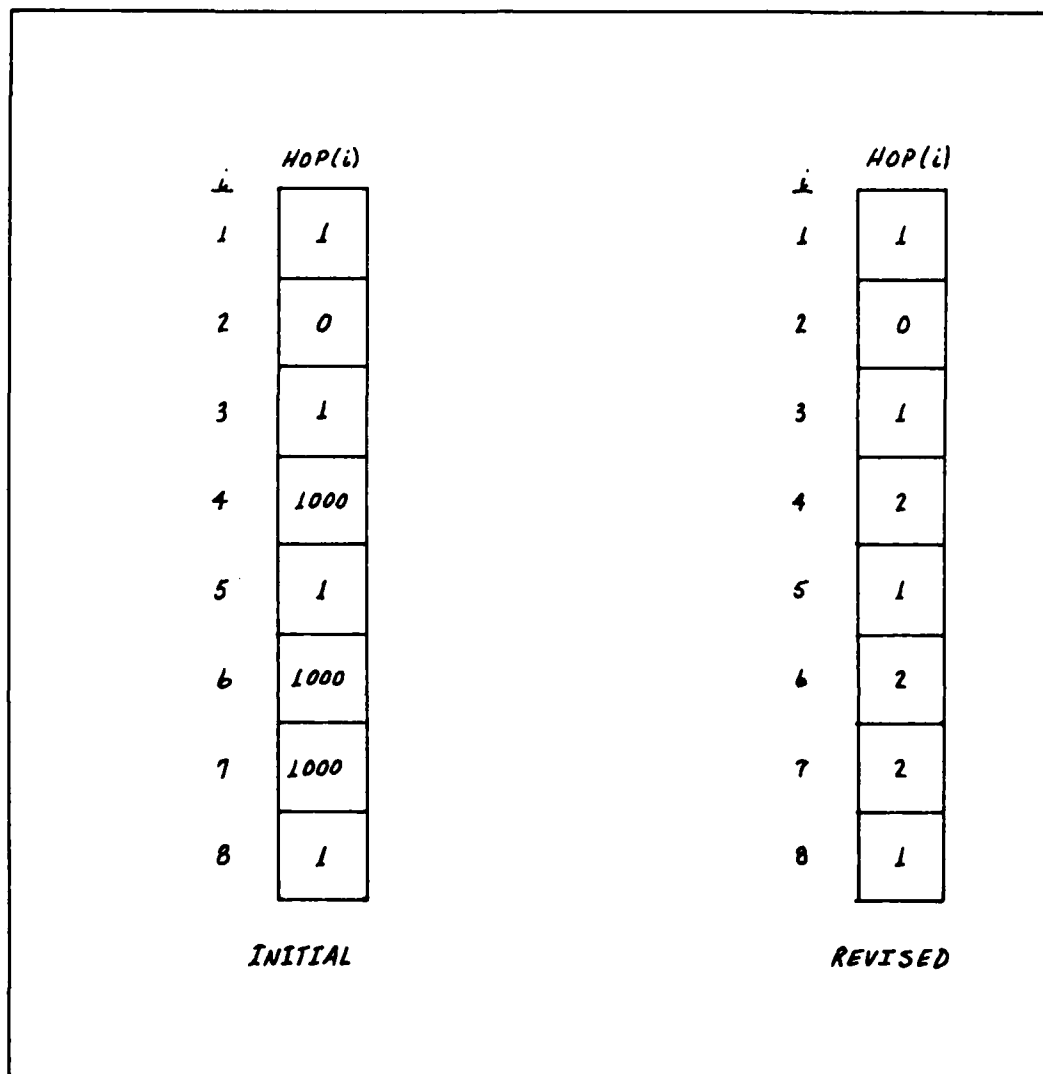


Fig 6. HOP Array for Andrews

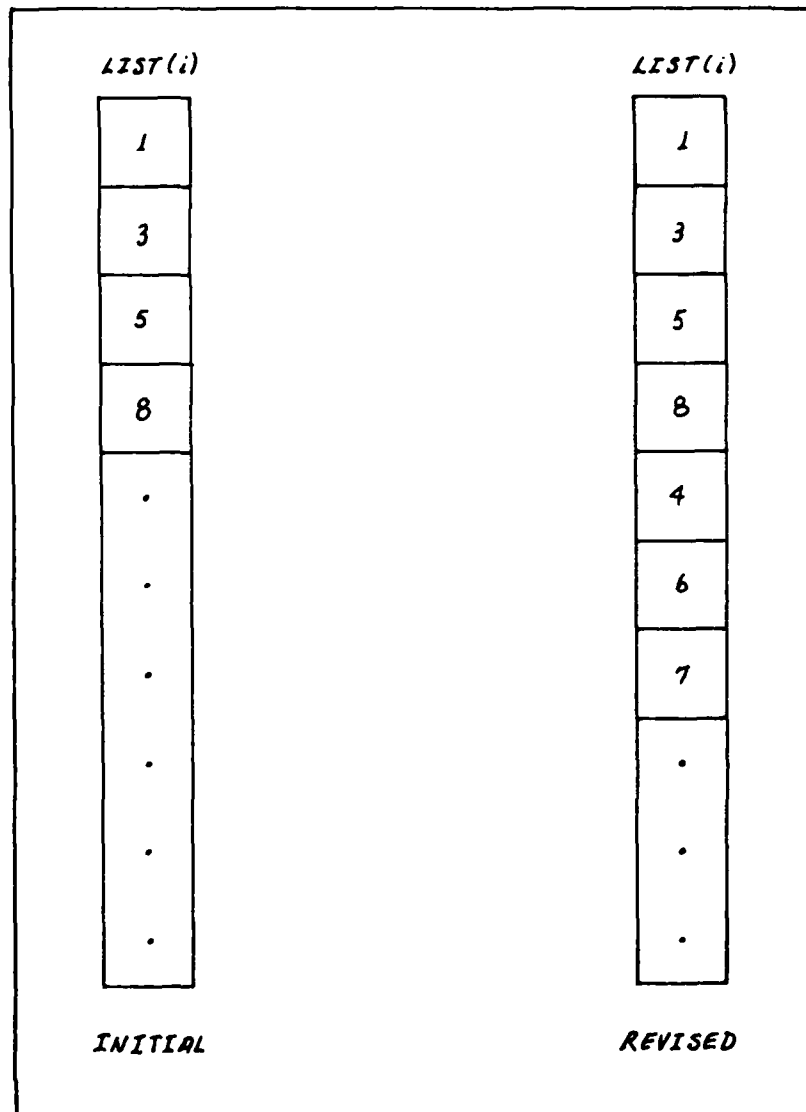


Fig 7. LIST Array for Andrews

$N \times NLN$  matrix, where again  $N$  is the number of nodes in the network and  $NLN$  is the number of links the node maintains. The entries reflect, for a given link, the number of hops (plus the congestion factor of .01 if congestion exists) required to reach a particular node. The table contains only minimum hop or minimum hop + 1 values. Minimum hop is defined as the least number of hops necessary for a packet to make, given the present node and the destination node. Minimum hop + 1 is just one more than minimum hop. Because a node is made up of multiple links, the number of hops a packet would be required to make on one link could be more or less than the packet would be required to make if routed on another link. If the number of hops required for a given link is more than the minimum hop + 1 count, that link's entry in the D table is the pseudo infinity value to preclude its usage in the routing calculations. Thus, each row of the D table gives the minimum hop/minimum congestion "distance" to a particular destination as a function of an outgoing link, such that only minimum hop and minimum hop + 1 paths have non-infinite values. Figure 8 shows the initial D table and the revised D table after initialization as calculated for the Andrews node. Note the initial values are similar to the constructs of the HOP array, that is, 1s and pseudo infinities. The source node entries are set to 0s for the set of outgoing links.

The SROUTE and ROUTE arrays are  $N \times 1$  arrays. An entry in either array represents the corresponding link to select,



$D(i, l)$

NODE(i)	i	OUTPUT LINK (l)			
		1	2	3	4
ALBANY	1	1	1000	1000	1000
ANDREWS	2	0	0	0	0
FT. DETRICK	3	1000	1	1000	1000
GENTILE	4	1000	1000	1000	1000
HANCOCK	5	1000	1000	1	1000
MCCLELLAN	6	1000	1000	1000	1000
NORTON	7	1000	1000	1000	1000
TENKER	8	1000	1000	1000	1

INITIAL D TABLE

$D(i, l)$

NODE(i)	i	OUTPUT LINK (l)			
		1	2	3	4
ALBANY	1	1	2	1000	2
ANDREWS	2	0	0	0	0
FT. DETRICK	3	2	1	2	2
GENTILE	4	3	2	2	2
HANCOCK	5	1000	2	1	1000
MCCLELLAN	6	3	3	3	2
NORTON	7	2	3	3	3
TENKER	8	2	2	1000	1

REVISED D TABLE

Fig 8. D Tables for Andrews

given the destination node of the packet. If the node processing the packet is the source node, the SROUTE array is used for link selection. If the node is a tandem node (a node other than the source node or the destination node), the ROUTE array is used. The rationale behind having two link routing arrays is based on the idea that under normal operating conditions, a packet will encounter at the most only one tandem node because of the network connectivity. The source node could have multiple "best" routes to select from to get the packet to its destination, but a tandem node, in general, would have only one "best" route which would be the link directly connecting it to the destination node.

#### Modules of the Algorithm

The hierarchical routing algorithm is made up of six modules. They are PAKROUTE, MESGEN, UPDATE, RTABLE, TTABLE, and MINHOP. The overall organization of the hierarchical routing algorithm is shown in Figure 9. The box representing the UPDATE module includes the RTABLE and TTABLE modules. The purpose of the AUTHENTICATION blocks is to insure that only actual nodes (not users) can send an authentic network status message. Otherwise a malfunctioning or malicious user could bring down the network by declaring all links down (Ref 8:A-3). The "Background Functions" and "Per Packet Functions" are discussed later in this chapter. Table I should be consulted for definition of

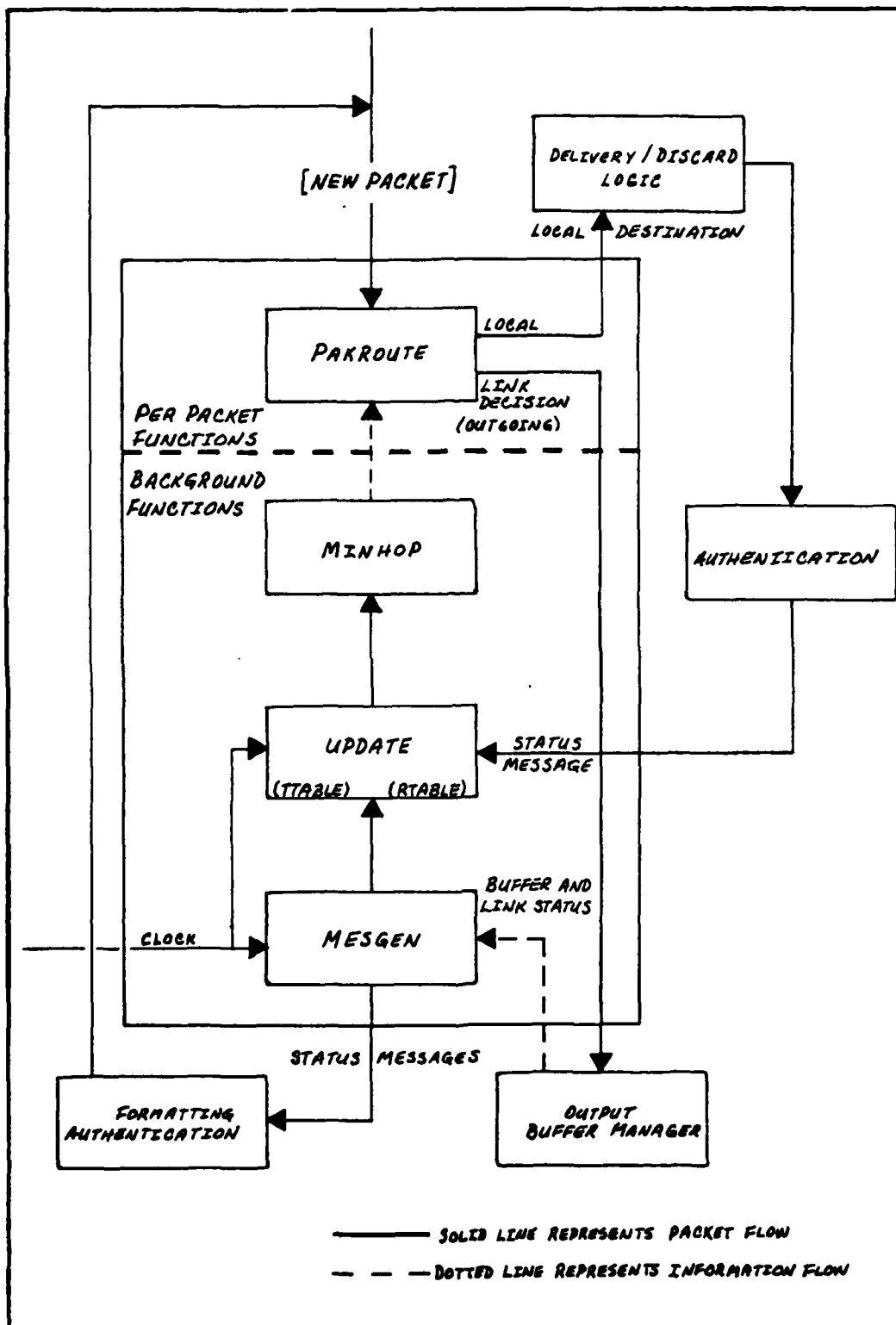


Fig 9. Overview of Routing Module

(Ref 8:A-4)

TABLE I  
Table of Variables

		Nominal Value
NN	Number of nodes in the network	8
NL	Number of one-way links in the network	32
NLN	Number of links outgoing from a particular node	3-5
INF	Pseudo infinity number > maximum number of hops in network	1000
LD(i,j)	Link distance matrix NN X NN	INF,1,1.01
SELF	Node self ID number	1-8
D(i,j)	Routing Distance table NN X NLN	-
P(i)	Connectivity Pointer array, (NN+1) X 1	-
NC (i)	Connectivity vector, NL X 1	-
ROUTE(i)	Tandem node routing array (outgoing link ID number indexed by destination node)	-
HOP(i)	Hop count array, NN X 1	-
LIST(i)	Connectivity set array, NL X 1	-
BNOM	Congestion weight	7
CNOM	Extra hop weight	7
STAT(i)	Node report status array, NN X 1	INF,1,1.01
T1	Congestion threshold (pkts per trunk)	7
T2	Saturation threshold (1,2,or 3 SCMs) total number of packets in input queues	7,18,20
TBUFF	Total number of packets in input queues	-
BUFF(i)	Number of packets queued on link i	-
LINES	Number of 56Kbps trunks serving link i	1-3
TABTICK	Time interval for running RTABLE	100 msec.

TABLE I (CONT'D)

		Nominal Value
FTICK	Time interval for "BAD" news detection in MESGEN	100 msec.
STICK	Time interval for "GOOD" news detection in MESGEN	400 msec.
SROUTE(i)	Source node routing array (outgoing link ID number indexed by destination node), NN X 1	-
ZONE	Flag denoting the status of the node on previous entry into MESGEN (0 - not saturated, 1 - saturated)	0,1
BUP(i)	Previous status of link array, NLN X 1 (1 - link up, 1000 - link down)	1,1000

variables used in the following figures as the algorithm is explained. The symbol Z in the figures is used to denote a sleep or idle point for a procedure.

PAKROUTE. The module executed for every packet processed is PAKROUTE, Figure 10. Depending on the destination of a packet, a link is selected from the SROUTE or ROUTE array (processing node is the source or tandem node of the packet). When the link contains only one trunk, the packet is queued for output on that trunk. If the selected link contains more than one trunk, the trunk with the shortest output queue is selected.

MESGEN. MESGEN is described in Figure 11. The prime function of MESGEN is the generation, when necessary, of status packets containing the node's current connectivity or congestion changes. A link is declared congested whenever the queued packets/trunk for a link exceeds a threshold T1. A node is considered to be saturated whenever the total of all input queued packets exceeds a threshold T2. It is entered periodically at two different time intervals, FTICK (100 milliseconds) and STICK (400 milliseconds). When entered as a result of FTICK, "bad news" is checked. Links are checked to see if any have gone down since the last FTICK entry. Congestion on the individual links is also checked. If none of the links have gone down, or there is no congestion on any of the links, then node saturation is checked. If the saturation threshold is equalled or sur-

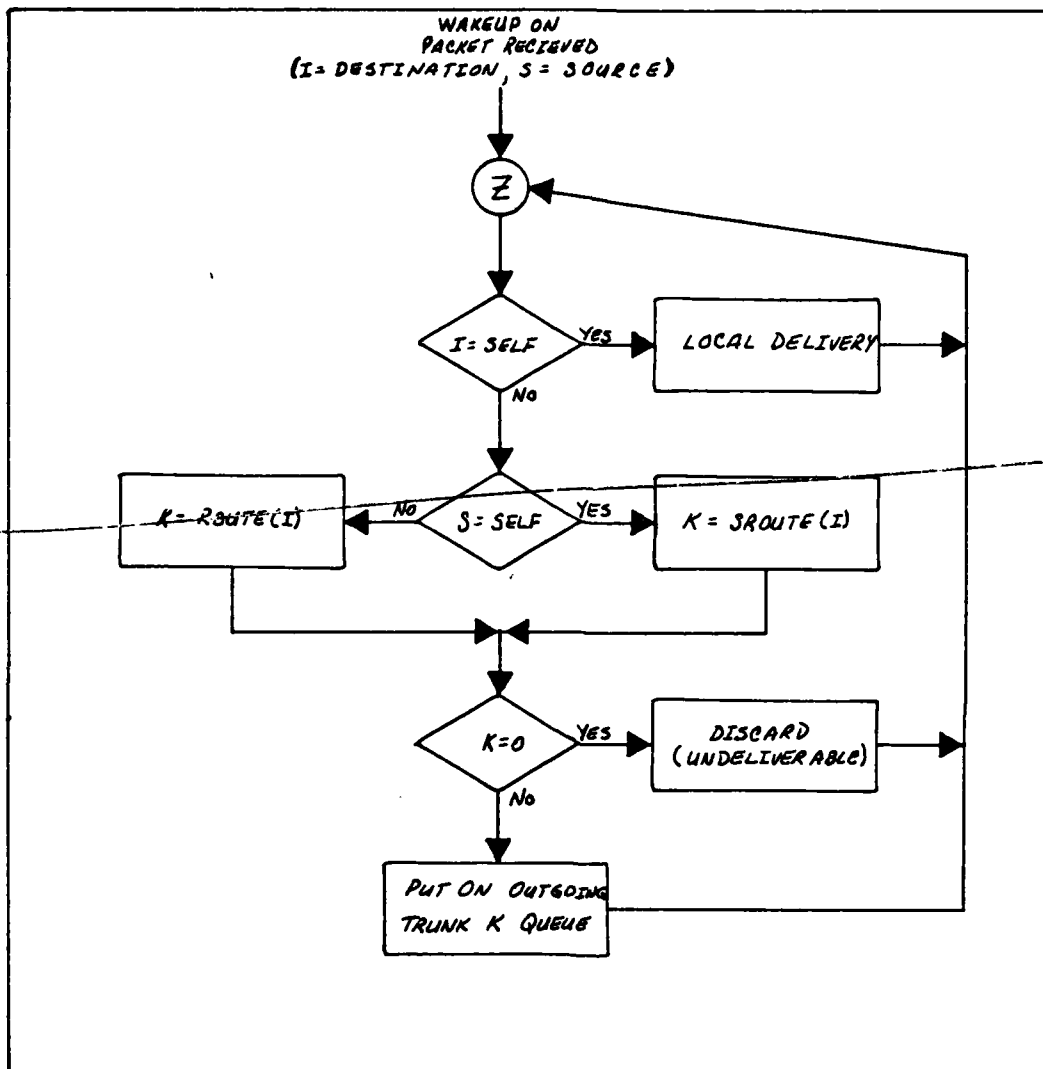


Fig 10. PAKROUTE

(Ref 8:A-5)

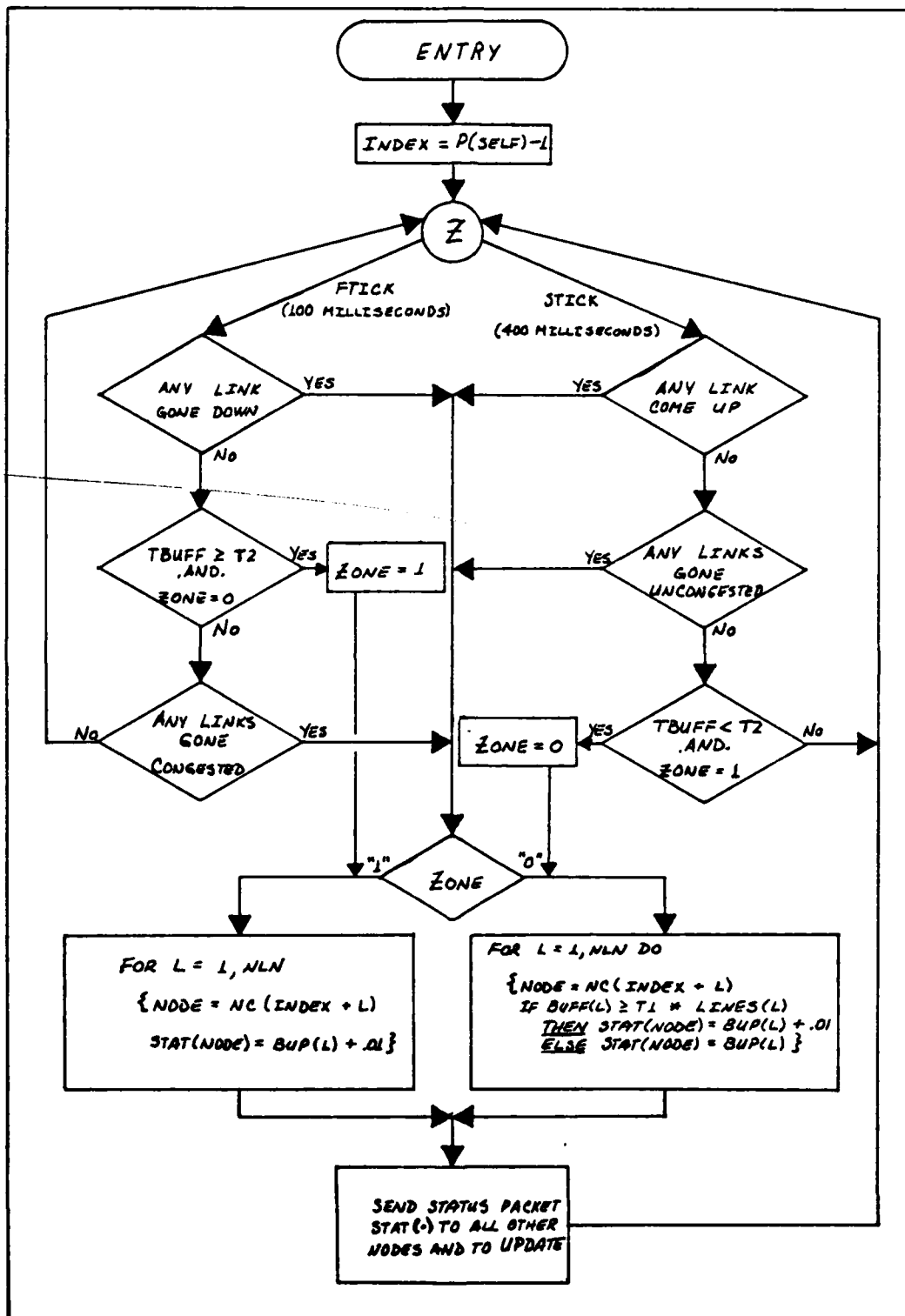


Fig 11. MESGEN



passed and the node was not saturated at the last FTICK entry (ZONE = 0), the node is considered saturated (Zone = 1), and a status message declaring all the node's links congested is formulated. If one or more links are down or congestion exists on one or more links a status packet is formulated selectively indicating congestion on the link or links in question. A copy of the status packet is sent to UPDATE for local processing. Copies of the status packet are also sent to other network nodes for their individual processing.

Upon entry because of the STICK time interval, "good news" is checked. Links are checked to see if any have come up as well as the absence of congestion on any of the links since the last STICK entry. If the status of the links are the same as the previous entry time interval, the saturation condition of the node is checked. If the node is currently not saturated and was saturated upon the last STICK entry into MESGEN (ZONE = 1), the node is considered not saturated (ZONE = 0). Even though the node is considered unsaturated, some links still might be congested, so they are checked individually for congestion in the event a link congestion status message is needed. If a link has come up or congestion no longer exists on a link, a status packet is generated by selectively checking each link. A copy of the status packet is sent to UPDATE for local processing. Additional copies are sent to other nodes for their processing.

Under entry into MESGEN because of FTICK or STICK, if

the processing gets down to check for the presence or absence of node saturation and the result of the check is negative, the module is exited with no further processing. Figure 11 is the corrected copy of the original module. An error was found during the code conversion to Fortran. The blocks explaining the actual entries into the status packet had been interchanged.

UPDATE. UPDATE is executed on each arrival of a status packet. As implemented in this simulation, for each pair of nodes, the LD table contains an entry indicating either the nodes are connected (1.00) and congested (1.01), or not connected (1000). Each arriving status message contains its source node's row of the LD table. The LD table is the table that is initially updated when a node receives a status message. The source node of the status packet encodes its corresponding row entries of the LD table and broadcasts it to the other nodes. Upon reception of the status message, the respective nodes compare the values in the status message against the appropriate row entry in their own LD table. If the values are the same the status message is discarded. If the values are different, the information in the status message replaces the corresponding row of the LD table. Figure 12 describes the procedure of UPDATE. It should be noted that the actual algorithm employs a 2-bit per entry scheme in the status message. In binary form 00 represents not connected, 01 represents connected but no congestion, and 11 represents connected and congestion exists. In the

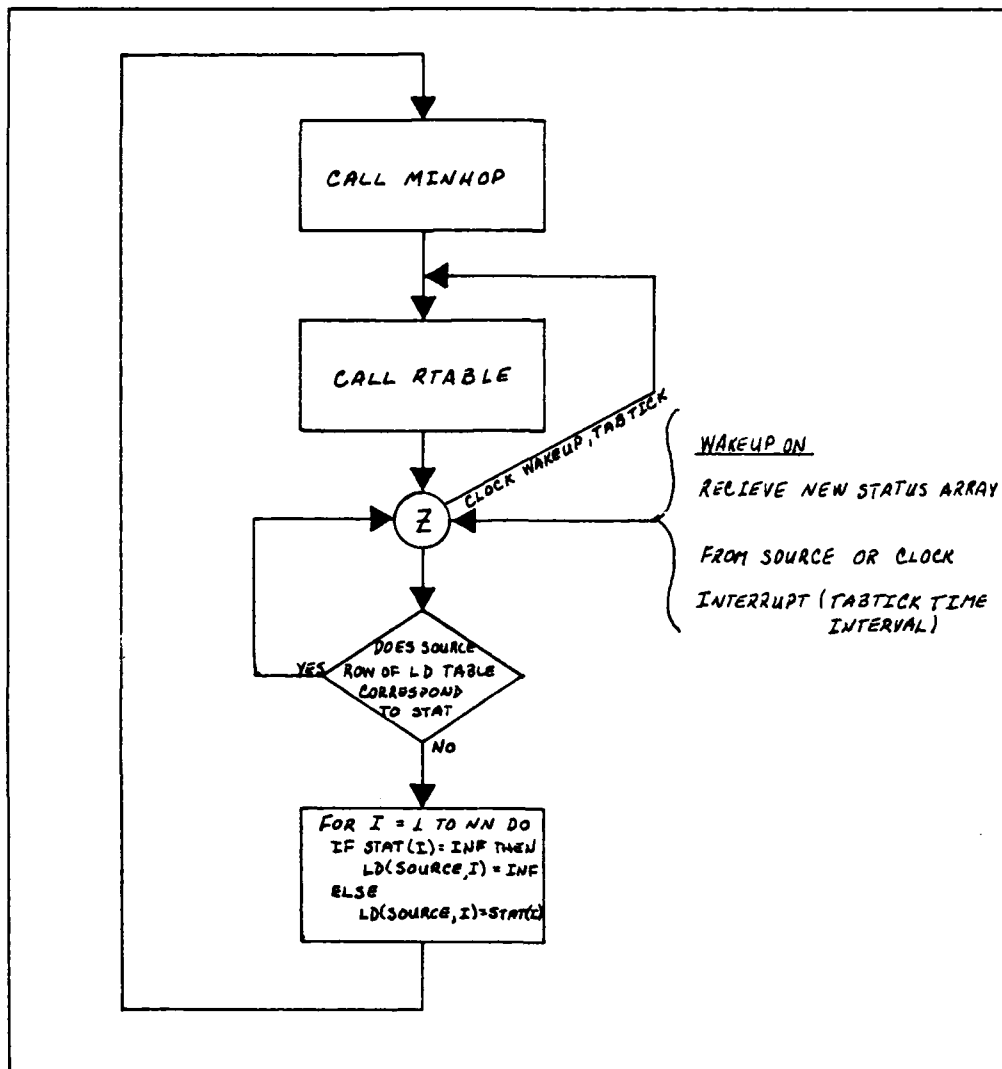


Fig 12. UPDATE

simulation, each entry in the status message uses one word to represent the above with entries of 1000, 1.00, and 1.01 respectively.

RTABLE. RTABLE is shown in Figure 13. The function of RTABLE is to update the source and tandem node routing arrays, SROUTE and ROUTE. Periodically, and after each execution of MINHOP, RTABLE is entered to recompute the routing arrays using the more current D table and local queue lengths.

TTABLE. TTABLE is the module used to build the trunk routing tables. The tables contain the designated trunks over which packets are to be routed, given a selected output link from SROUTE or ROUTE. The implementation of the routing algorithm in the simulation model did not execute TTABLE. Instead, the length of the queues of each trunk in the selected output link is scanned to select the trunk with the shortest queue. That same procedure is used in the execution of TTABLE in any event, only the extra code to actually build and maintain the trunk routing tables TTABLE was not used.

MINHOP. MINHOP calculates the distance table (D table) indexed by outgoing links and final destination node. Table entries encode (conditioned on outgoing links and destination node) the minimum hop count and minimum congestion along minimum hop paths. Entries are pseudo infinite values if the hop count (for outgoing links and destination

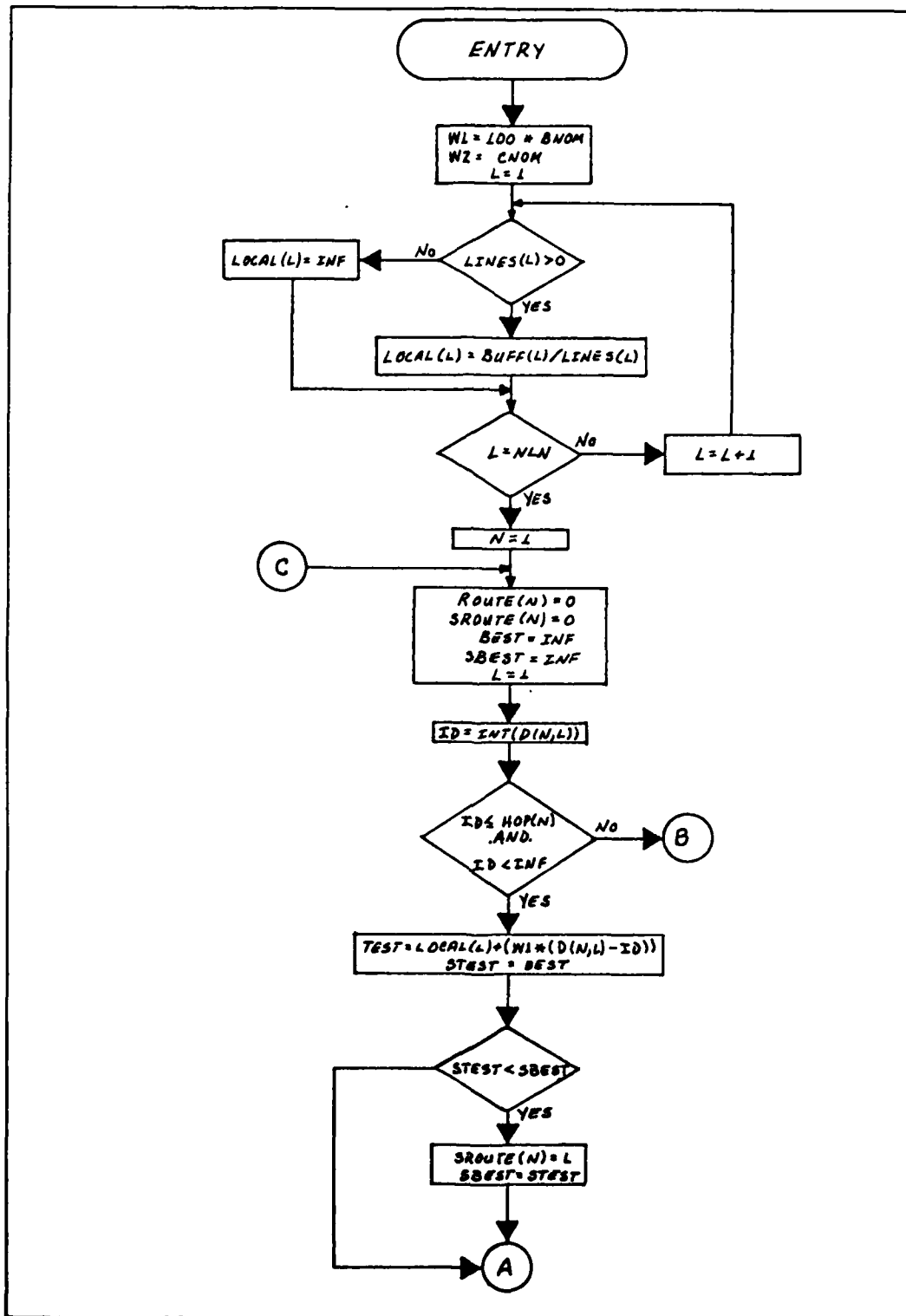


Fig 13. RTABLE

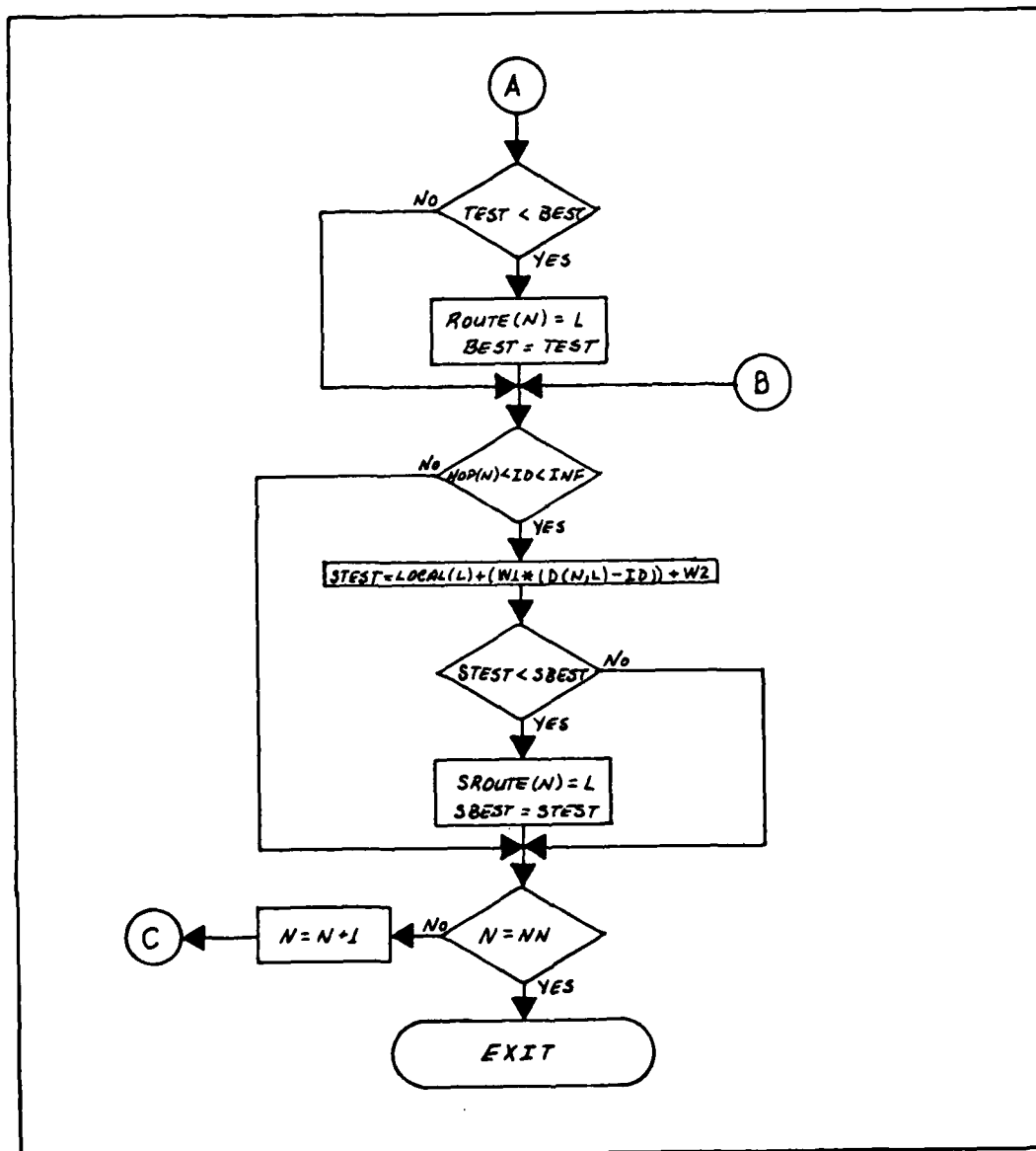


Fig 13. (CONT'D)

node) exceeds the overall (for destination node) minimum hop count plus one (Ref 7:7).

MINHOP is the heart of the routing algorithm. It can be described in four procedures: (1) CONNECTIVITY, (2) INITIALIZE\_D, (3) COMPUTE\_D, and (4) CONN\_CHANGE. CONNECTIVITY, Figure 14, converts the connectivity information in the link distance (LD) table into more convenient forms of the P and NC arrays. It is entered only once, during initialization. INITIALIZE\_D, Figure 15, initializes the D table from the values of the LD table, P and NC arrays. COMPUTE\_D, Figure 16, executes a search process to fill out the rest of the D table. The above two procedures are entered if, after a status packet is received, an update to the D table is required. In the interest of clarification, any further references to MINHOP will be synonymous with the execution of INITIALIZE\_D and COMPUTE\_D. CONN\_CHANGE, Figure 17, updates the NC array to reflect link failure or repair. In addition, it updates the LD table which stores the link distance and congestion information (Ref 7:27). CONN\_CHANGE is entered when the status of a link changes.

#### Background Functions

Figure 18 pictorially describes the functions performed by the algorithm in the Background mode. The actuation of a particular procedure is either periodical or event driven. In the Background mode the tasks MINHOP, RTABLE, TTABLE, UPDATE, and MESGEN are executed periodically.

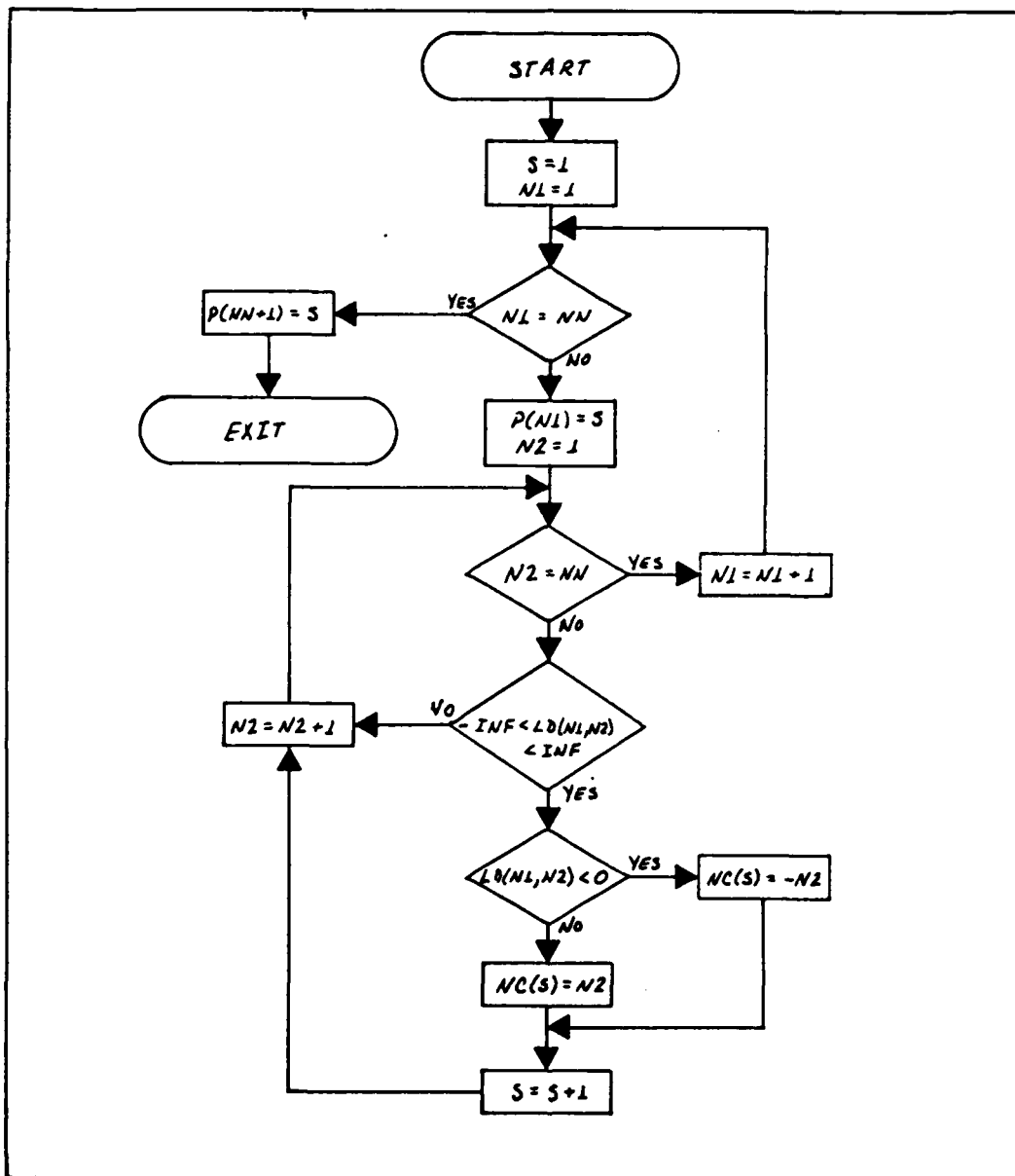


Fig 14. CONNECTIVITY



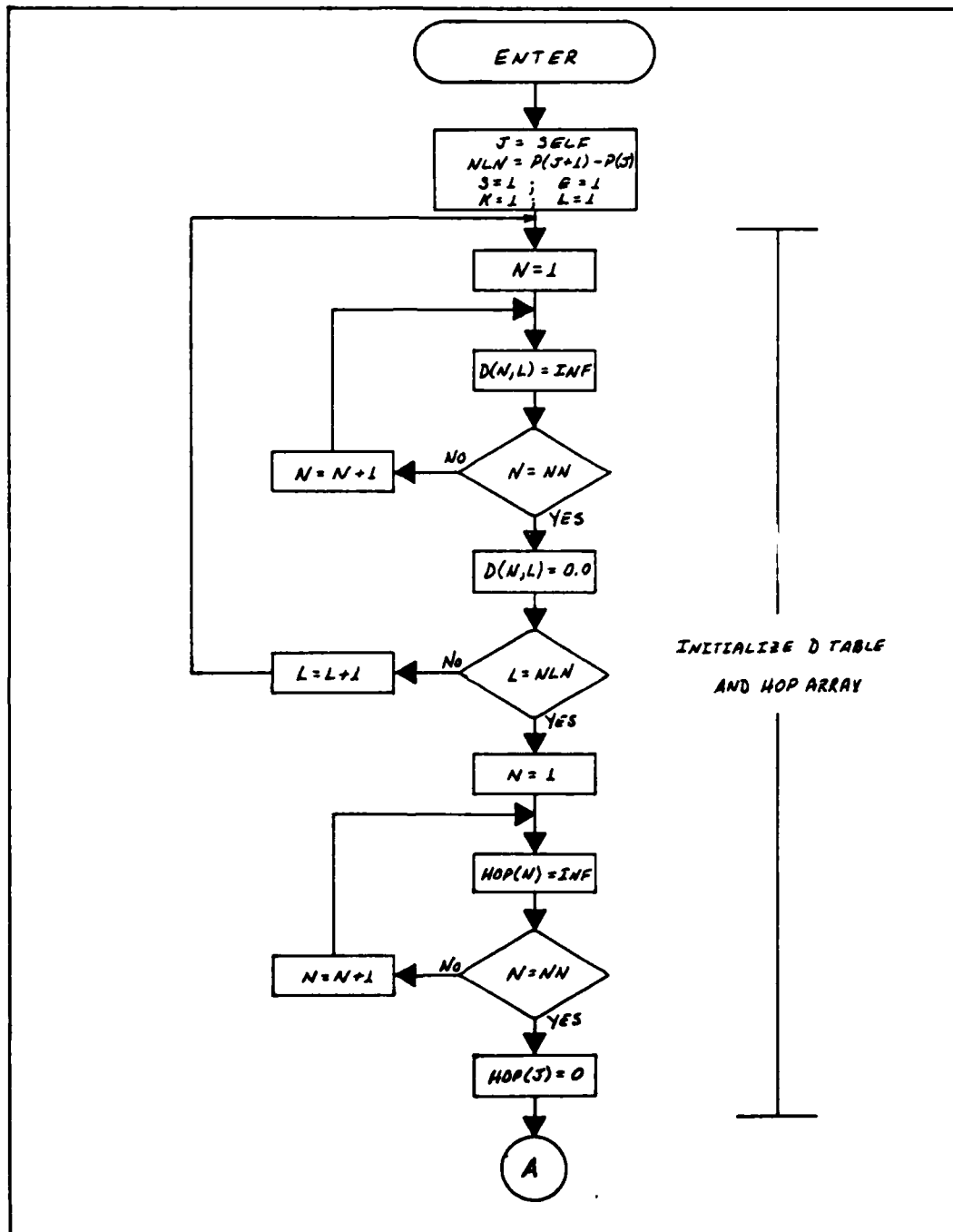


Fig 15. INITIALIZE\_D

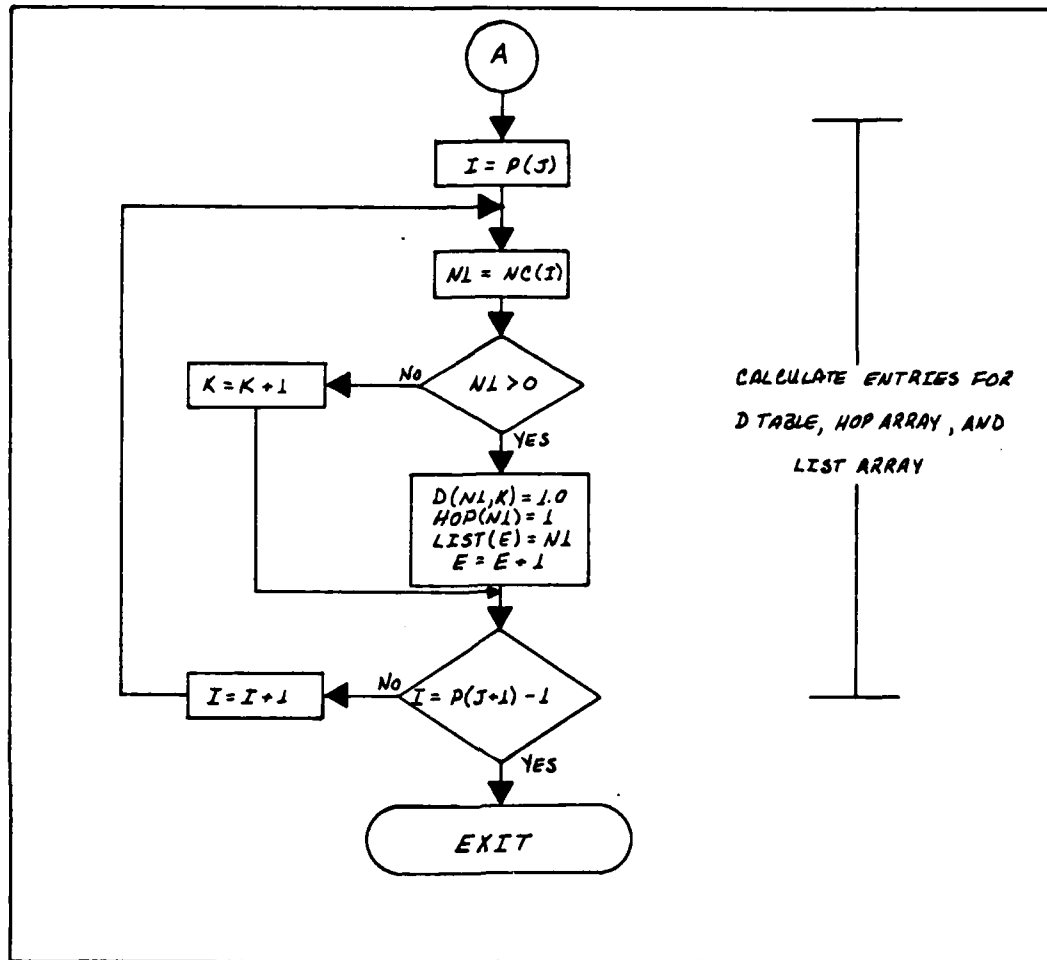


Fig 15. (CONT'D)

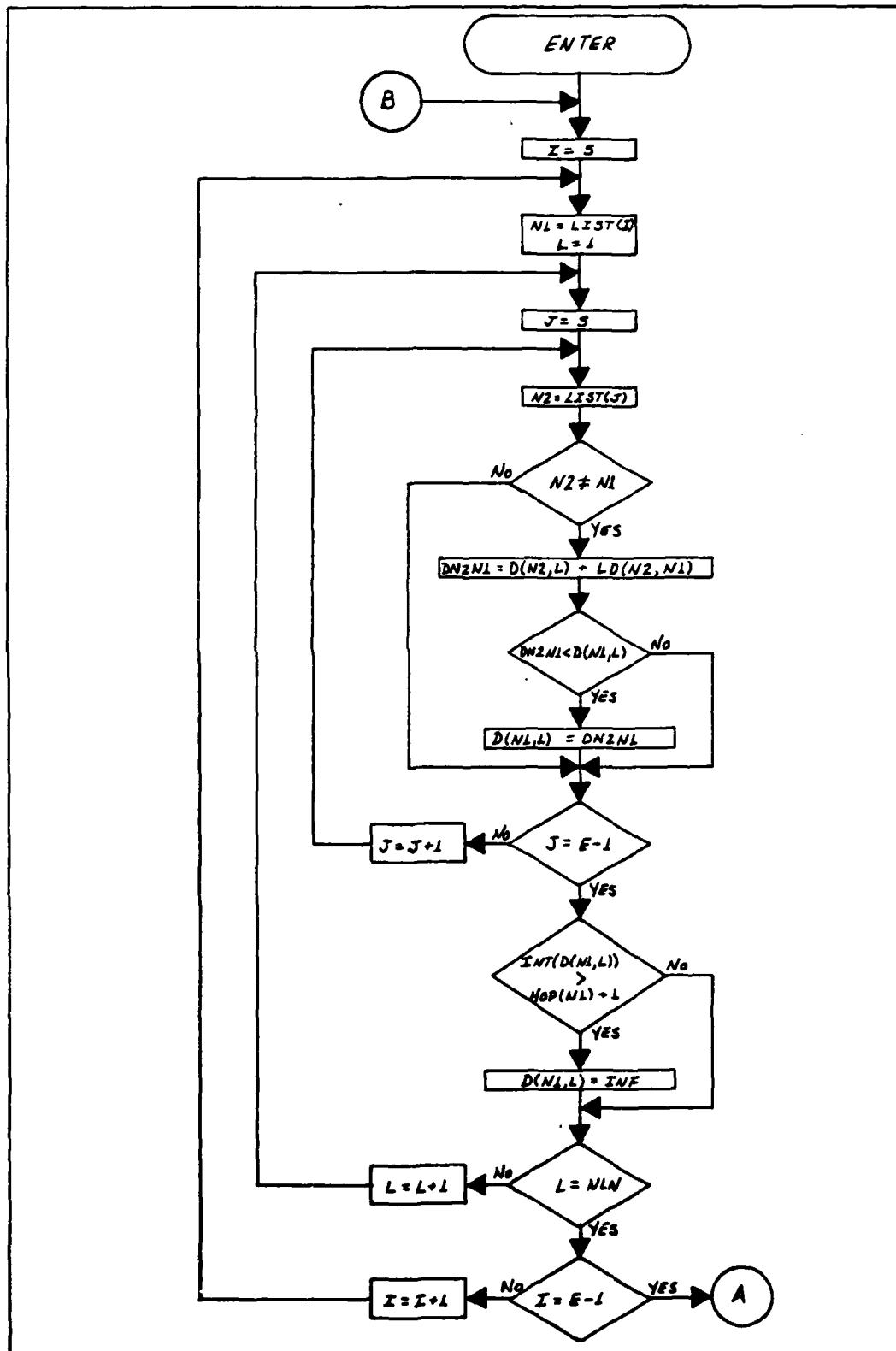


Fig 16. COMPUTE\_D

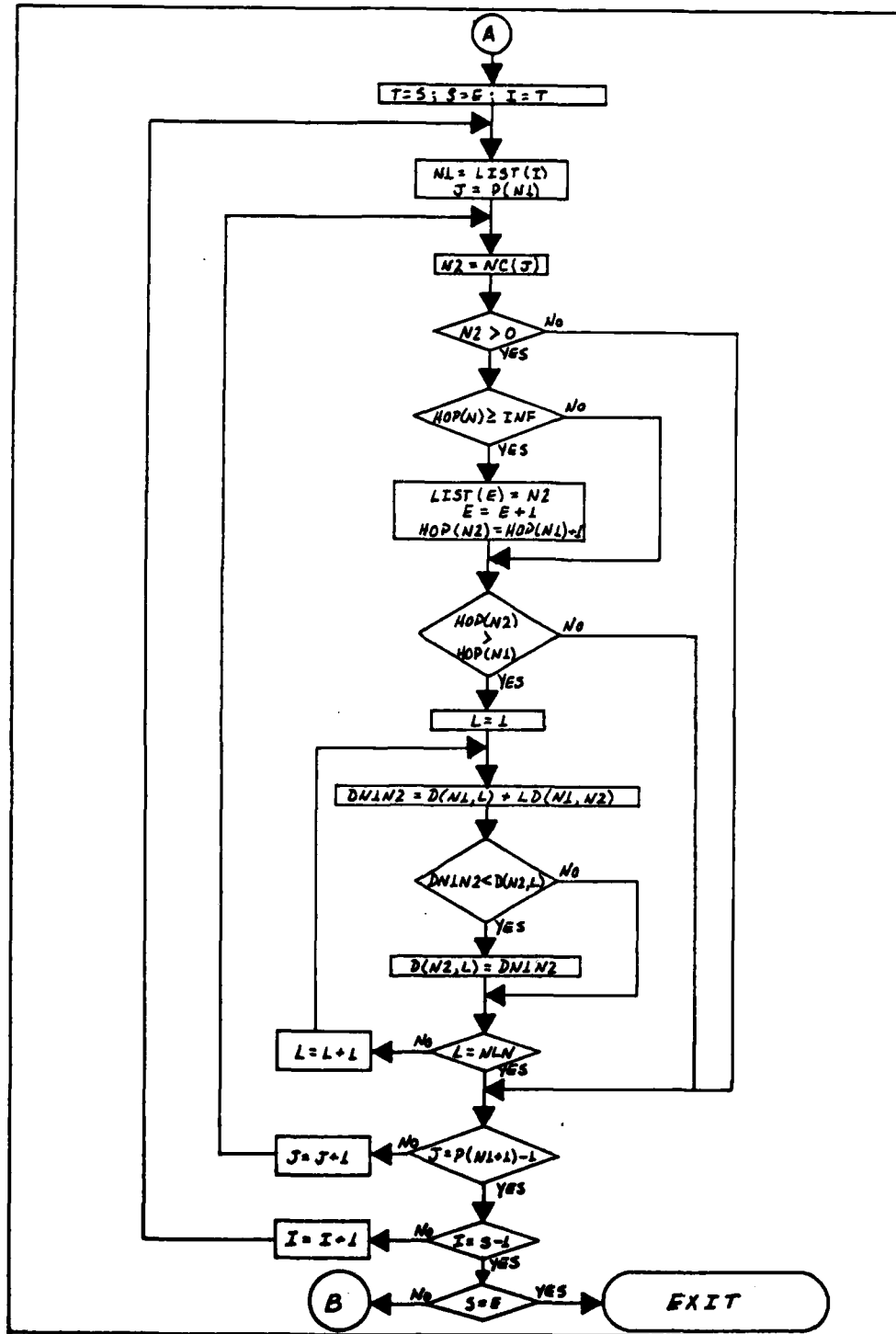


Fig 16. (CONT'D)

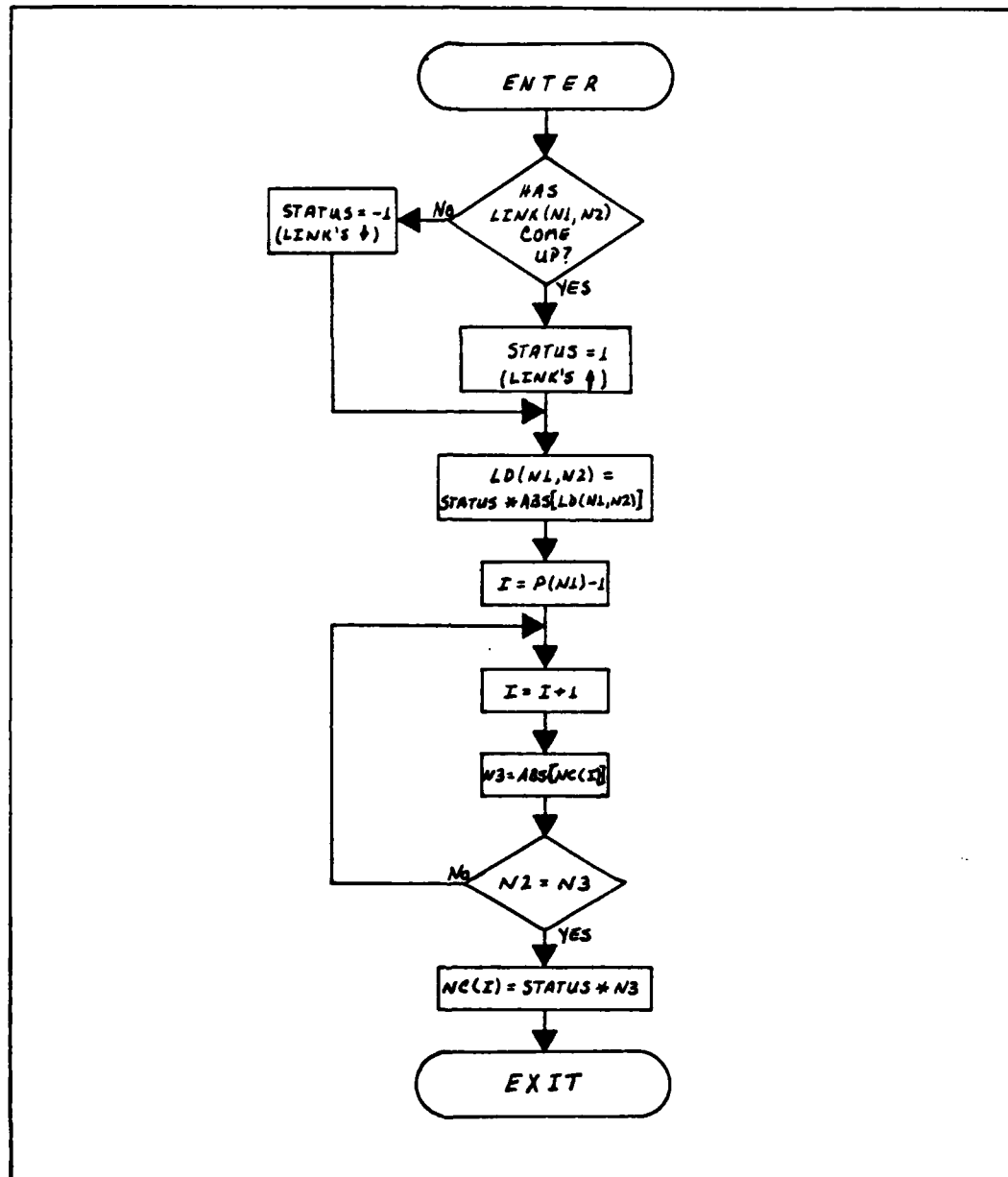


Fig 17. CONN\_CHANGE

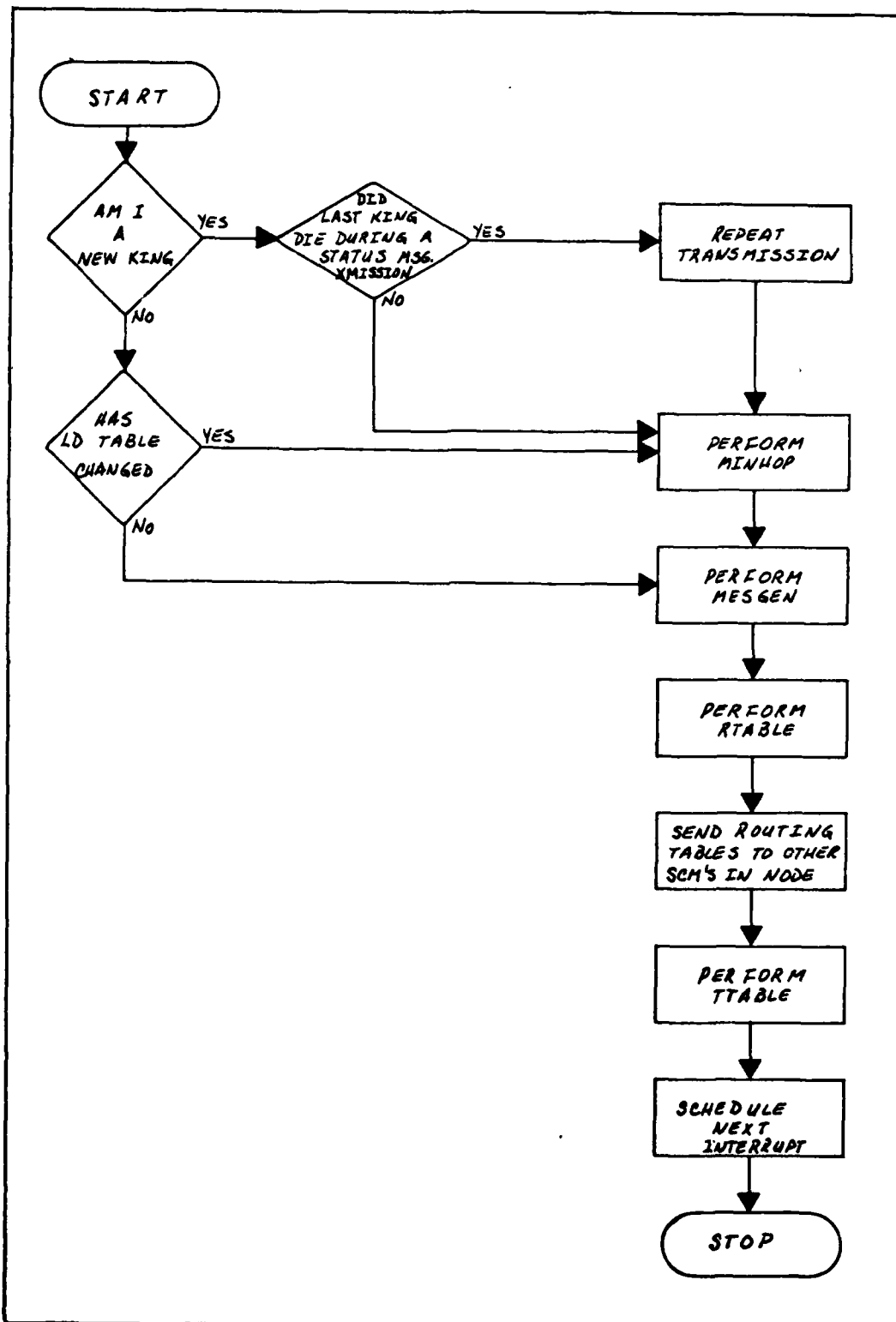


Fig 18. Logical Order of Background Tasks

(Ref 7:19)

If it is time to execute one of the above modules, the processor enters a multi-processing mode where it can process a packet and execute the appropriate module at the same time.

The King determination process was not incorporated into the algorithm as simulated, but will be incorporated into the ON-LINE version of the algorithm. The background tasks of MINHOP, RTABLE, TTABLE, UPDATE and MESGEN need to be done only at one SCM per node. One SCM will thus be designated as the "King", and have responsibility for these tasks (Ref 7:4). The King will be periodically determined in a distributed fashion as the SCM having the minimal load. In the case of a failure of a King processor the next King determination will reestablish order (Ref 7:4). King determination will be interrupt initiated and will be reassigned only upon failure of the current King processor.

To supply the King with the necessary queue information, the SCMs will periodically exchange information vectors containing (Ref 7:4):

- (1) The processor utilization during a specified time interval or other estimates of the current processor load.
- (2) The input queue size for processing at the SCM.
- (3) The output queue sizes for each of the trunks emanating from the SCM having the minimum processor utilization.

Once the King has been established, the queue sizes for trunks are consolidated as queue sizes for links and

the King assumes command of the execution of MINHOP, RTABLE, UPDATE and MESGEN. The "Coronation Process" must be executed identically at multiple-SCM nodes in order for each SCM to arrive at the same SCM to choose as King. At any time, the King might die (at the very least hardware errors are very unpredictable), therefore, the other SCMs at a node must maintain the information necessary to assume command.

#### Per Packet Functions

Figure 19 shows a schematic view of the per packet routing tasks. The routing tables produced by RTABLE are used to determine which outgoing link, if any, is to be used. PAKROUTE splits the packets into three broad categories: (1) Packets destined for other nodes, (2) packets destined for the present node, and (3) undeliverable packets (due to hardware failures separating part of the network, etc.) (Ref 4:40). The per packet function mode was not entirely incorporated into the algorithm as exercised in the simulation. The use of the Processor Communications Link (PCL) was not implemented. When a packet enters a node, PAKROUTE is executed by the individual processor processing the packet. As a result of PAKROUTE, the packet is then queued directly to whichever trunk was selected. The functions as portrayed in Figure 19 will be implemented into the ON-LINE version of the algorithm.

As explained in Chapter VII, the hierarchical routing



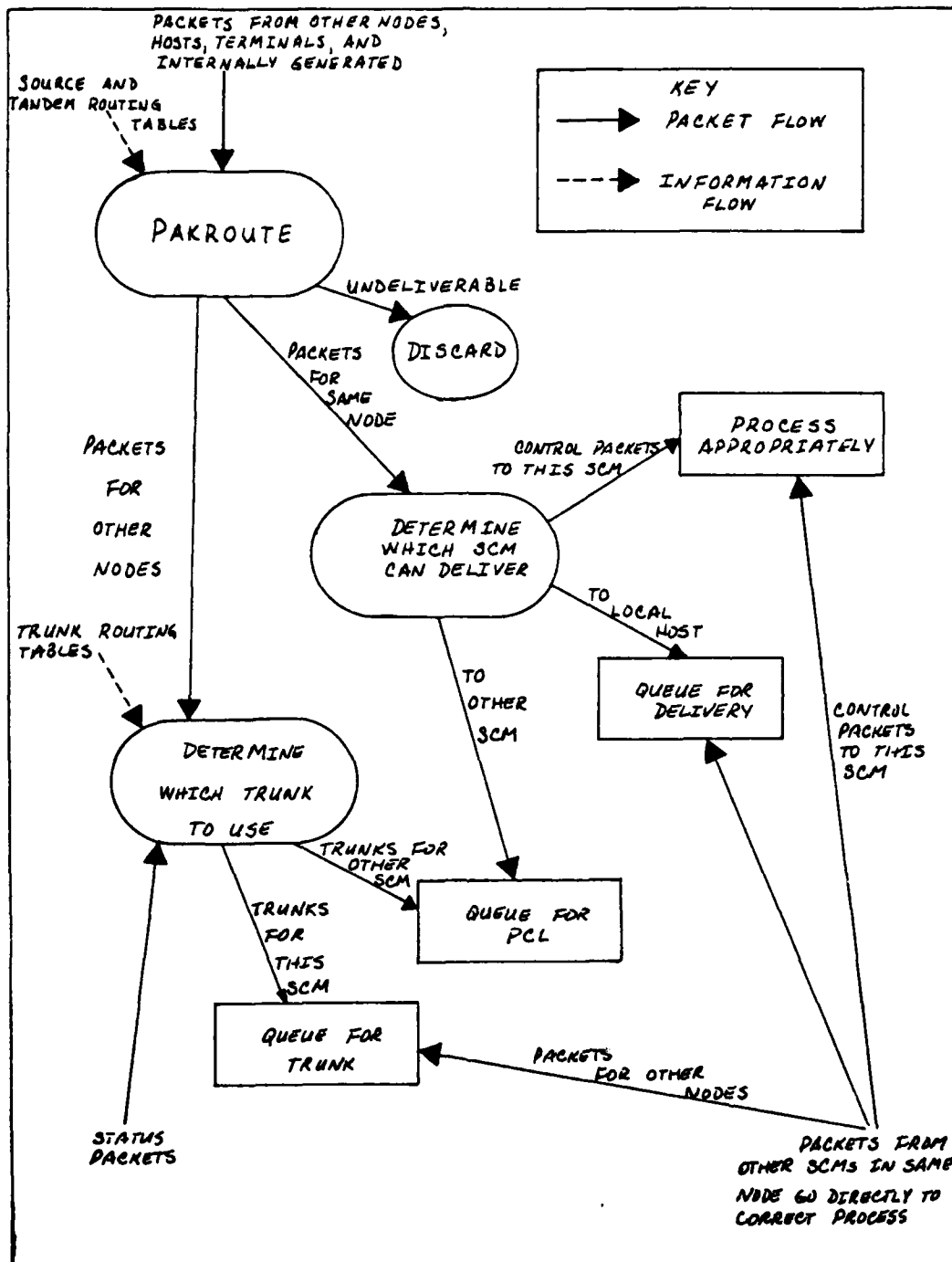


Fig 19. Schematic View of Per Packet Tasks

algorithm was written in the high order programming language FORTRAN IV. The rest of the simulation model was coded in the simulation language, General Purpose Simulation System (GPSS). The following chapter provides a description of GPSS along with a detailed description of the simulation code for one of the nodes in the network.

## V GPSS SIMULATION MODEL

### Language Description

GPSS (General Purpose Simulation System) was developed in 1961 by Geoffrey Gordon. The primary purpose in developing such a simulation language was to develop a simulation tool for use by systems analysts who were not adept in computer programming. Because of the universality of flow charts, GPSS was structured as a block-oriented language. This idea allows the modeler to construct the model of a system using blocks which are connected together to form a network. The blocks are connected together in the same sequence of events as occur in the system being modeled (Ref 9). A set of unique block types are defined, each corresponding to a basic system action. In this way, system actions and the times at which those actions occur are modeled by drawing a block diagram of the system and inserting the appropriate block instruction to accomplish the desired action. The components of the model on which the system acts are called transactions. They in turn, model the dynamic structures of the system. Transactions move through the block structured program, being created, modified, delayed, transferred, or destroyed as required.

Although GPSS was developed to model and simulate a vast scope of various real world physical and abstract systems, it particularly lends itself to the discrete-event simulation of queueing systems. The natural order of en-

titles, instructions, model description and internal organization of the simulator allows a queueing model to be developed, coded, and simulated almost effortlessly.

Transactions, as described earlier, are the principle source of traffic flowing through the model. They represent some operation of the system being simulated but the modeler defines what those operations are. Transactions move from block to block in a manner similar to units of traffic in the real system they represent. Each movement is considered to be an event that is due to occur at some point in time. GPSS automatically maintains a sequential record of the events which are to occur. In those cases when events (such as attempting to seize a facility which is already in use) cannot take place at the instant in time when they were originally scheduled, the transaction ceases to move until the event can take place. During a transaction's scheduled movement, it moves through as many blocks as it can before being denied entry to a block. The program maintains the status of the condition causing the transaction to be delayed. As soon as the condition changes, the transaction proceeds in the appropriate sequence with respect to the other transactions, that is, the transactions scheduled to move first do so. Associated with each transaction are attributes called parameters. These parameters enable the modeler to keep track of, collect, and modify dynamically those characteristics which are of particular interest to the modeler about that particular transaction.

The GPSS model is made up of blocks correlating to the block diagram constructed to define the modeled system. Each block can be thought of as a small subroutine or macro instruction. Each block is associated with information falling into three categories; location, operation, and operands (Ref 10). Every block occupies a specific location in the block diagram. The first block in a model occupies location 1, the second block occupies location 2, and so on. The GPSS Processor automatically assigns a location number to each block unless the user option is utilized in which the modeler labels a particular block with a name of his own choice containing from three to five alphanumeric characters, the first three of which must be alpha. The block's number, or label, can be referenced in later blocks. The "operation" of a block is a "verb" suggestive of the task the block accomplishes. Each block type is characterized by its own predefined verb, that is, GENERATE, ASSIGN, QUEUE, SEIZE, and TRANSFER just to name a few. A block has operands whose values identify the specific actions the block is to take. The operands can be conveniently thought of as arguments used in subroutine calls. Four basic types of events occur with a block: (1) creation or destruction of a transaction, (2) alteration of a numerical attribute of a transaction, (3) delay of a transaction for some specified amount of time, and (4) alteration of the transaction flow through the model.

There are four categories of block types. They are transaction-oriented, transaction flow-modifying, equipment-oriented, and statistical. Transaction-oriented blocks deal with the transaction itself. A transaction can be created as in the GENERATE block, destroyed as in the TERMINATE block, delayed as in the ADVANCE block, have its parameters modified as in the ASSIGN block, or have its priority changed as in the PRIORITY block. Flow-modifying blocks modify the flow of transactions through the model. The TRANSFER block is generally used to direct a transaction to a nonsequential next block. The LOOP block causes the transaction to loop through a specified set of blocks until a condition is met or satisfied. The TEST block allows transfer of a transaction upon testing a specified condition which is the result of an algebraic comparison between two operands of the block. Equipment-oriented blocks define how a transaction controls the simulated device, facility, equipment, process or other entity the transaction is currently using. The SEIZE block indicates the use of a facility by the entering transaction so that the facility remains in use until the seizing transaction releases the facility by entering a RELEASE block. When a transaction with high priority enters a PREEMPT block, it suspends the progress of the lower priority transaction which had previously seized the facility. The higher priority transaction gains control of the facility and when it is through with the facility,

returns control back to the preempted transaction by entering a RETURN block. If some type of storage entity is being modeled, a transaction entering an ENTER block obtains a specified number of units of the storage entity. When the used storage is to be returned, the transaction enters a LEAVE block. Statistical block types cause specified statistics to be gathered and stored pertaining to the desired program entity. When a transaction enters a QUEUE block, queue statistics on that transaction such as the number of transactions currently in the queue, maximum number of transactions in the queue at any one time, average time for a transaction to spend in the queue, and the number of transactions that incurred no wait time spent in the queue are activated. The DEPART block removes the transaction from the queue and causes the actual statistics discussed above to be gathered and stored for later output at the end of the simulation. The blocks discussed above are by no means all the blocks available but do lay the fundamental groundwork needed to understand the simulation model to be discussed shortly.

Output from the simulation includes statistics on the queues, facilities, storages, specifically defined program entities called SAVEVALUES, and any tabular data generated by transactions entering a TABULATE block.

The following section describes the basic program used to simulate the AUTODIN II simulation model.

### Program Description

Two versions of the simulation program were used. Chapter VI explains the specific details for the Baseline or control program using fixed routing. Chapter VII explains the incorporation of the hierarchical routing algorithm into the simulation model.

The simulation program models the eight node AUTODIN II communication computer network. The eight nodes are Albany, Andrews, Ft. Detrick, Gentile, Hancock, McClellan, Norton, and Tinker. Figure 20 shows how the nodes are connected together, the number of SCMs at each node, and how the trunks are connected to the SCMs at the nodes. In general, the processing and routing of packets at each node is the same. The simulation program must generate packets (simulating packets received from locally connected Host computers and terminals), assign the appropriate priority, length, source and destination node identification number to each packet, and queue and transmit the packet to the appropriate node (which will either be the destination node or an intermediate tandem node). Figure 21 is typical of the program block diagram for each node. The number of SCM processors, queues, and trunks a node has is peculiar to each node. In this example, the block diagram is shown for the Andrews node in the Baseline simulation model. The packets are generated with some mean interarrival rate peculiar to each node and are exponentially distributed. In Figure 21, the GENERATE



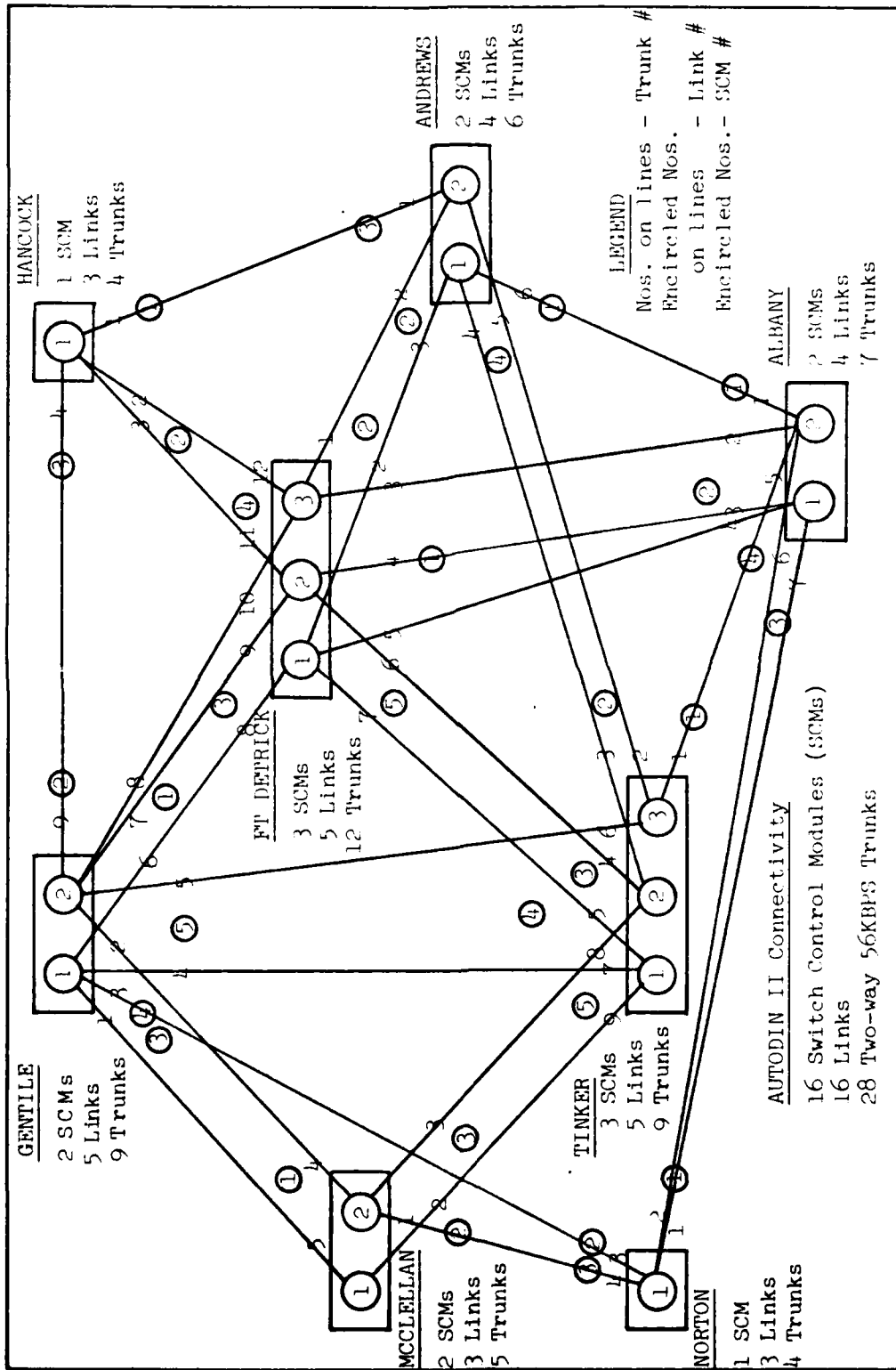


Fig 20. Network Layout of AUTODIN II

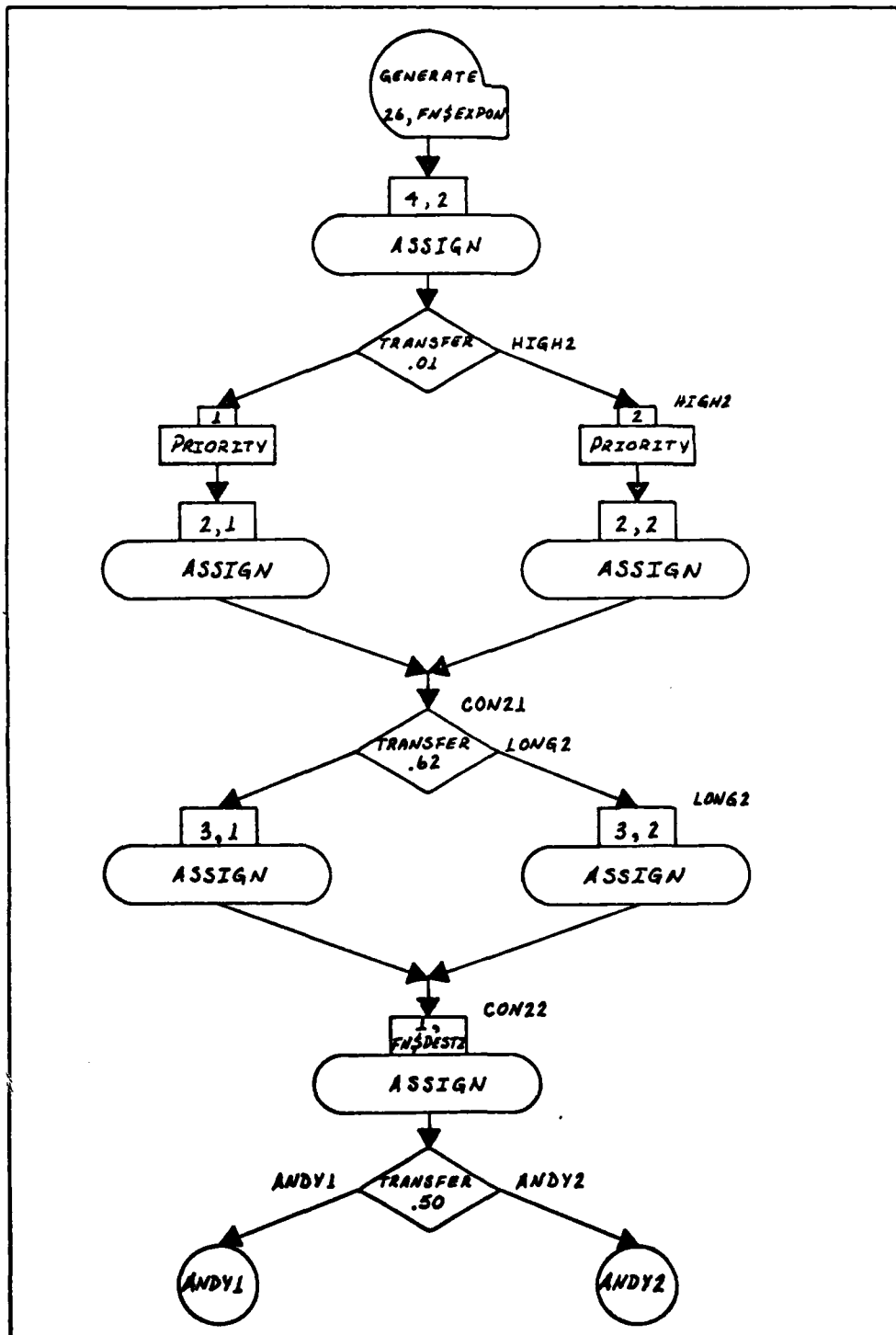


Fig 21. GPSS Block Diagram for Andrews

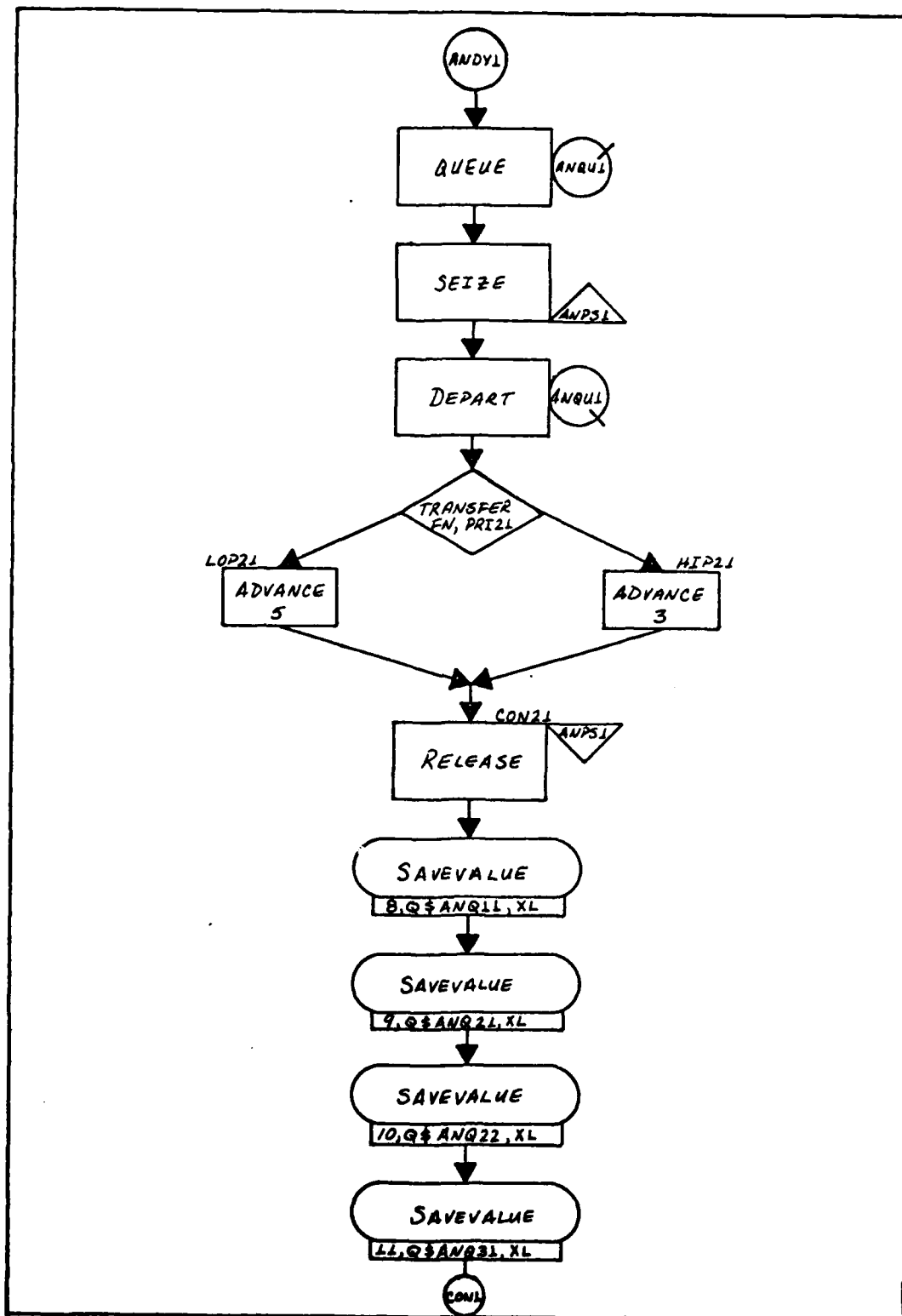


Fig 21. (CONT'D)

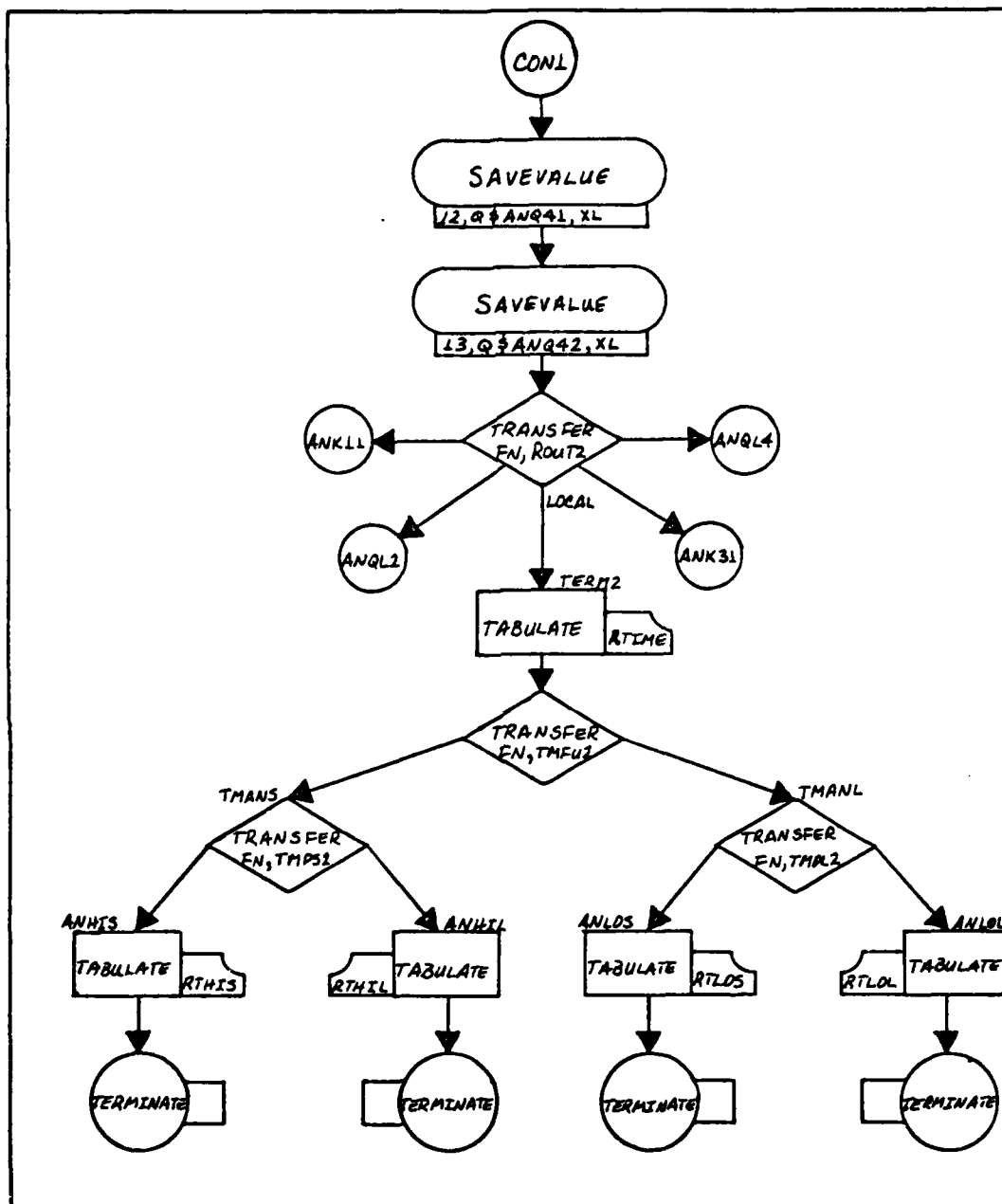


Fig 21. (CONT'D)

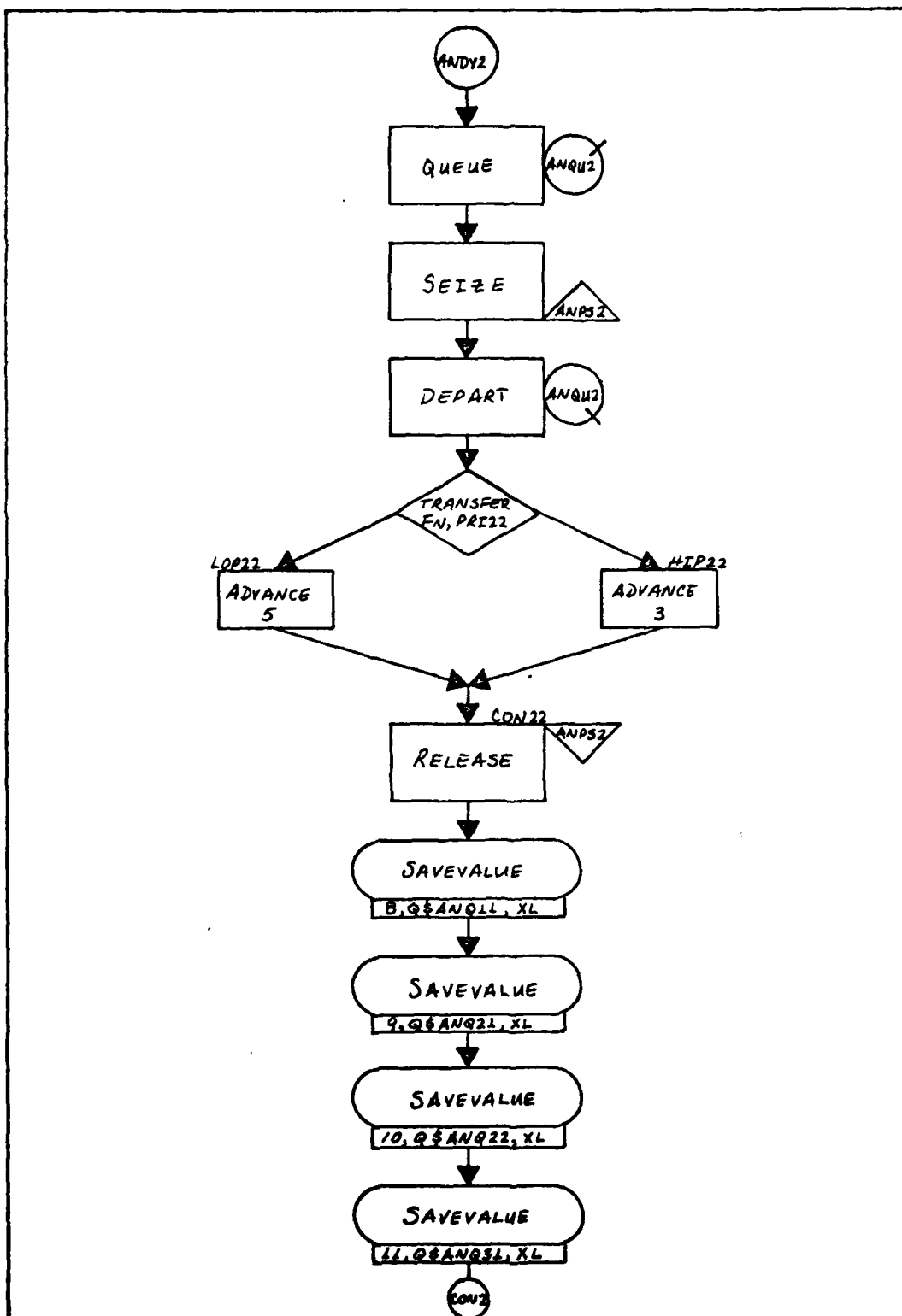


Fig 21. (CONT'D)

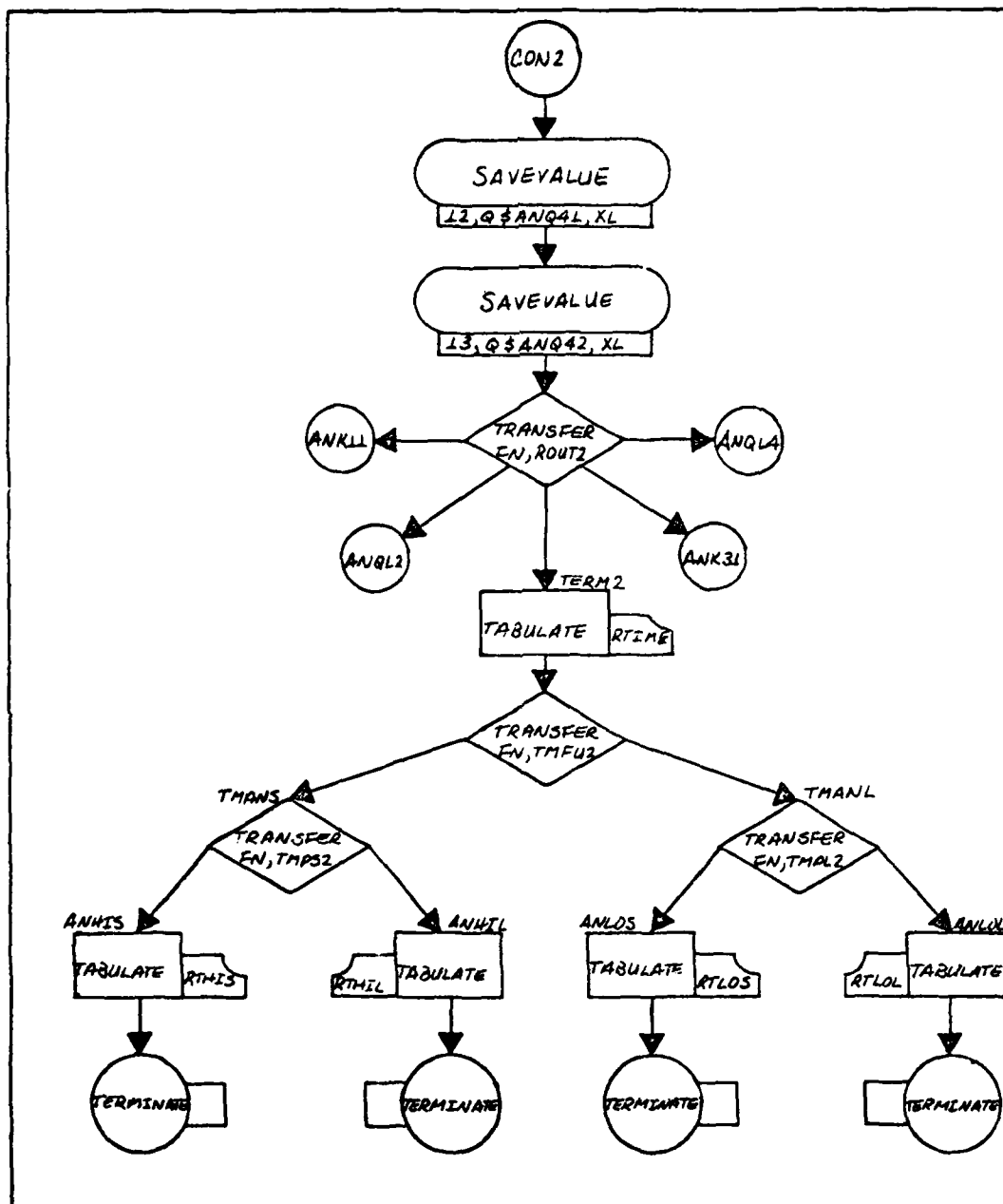


Fig 21. (CONT'D)

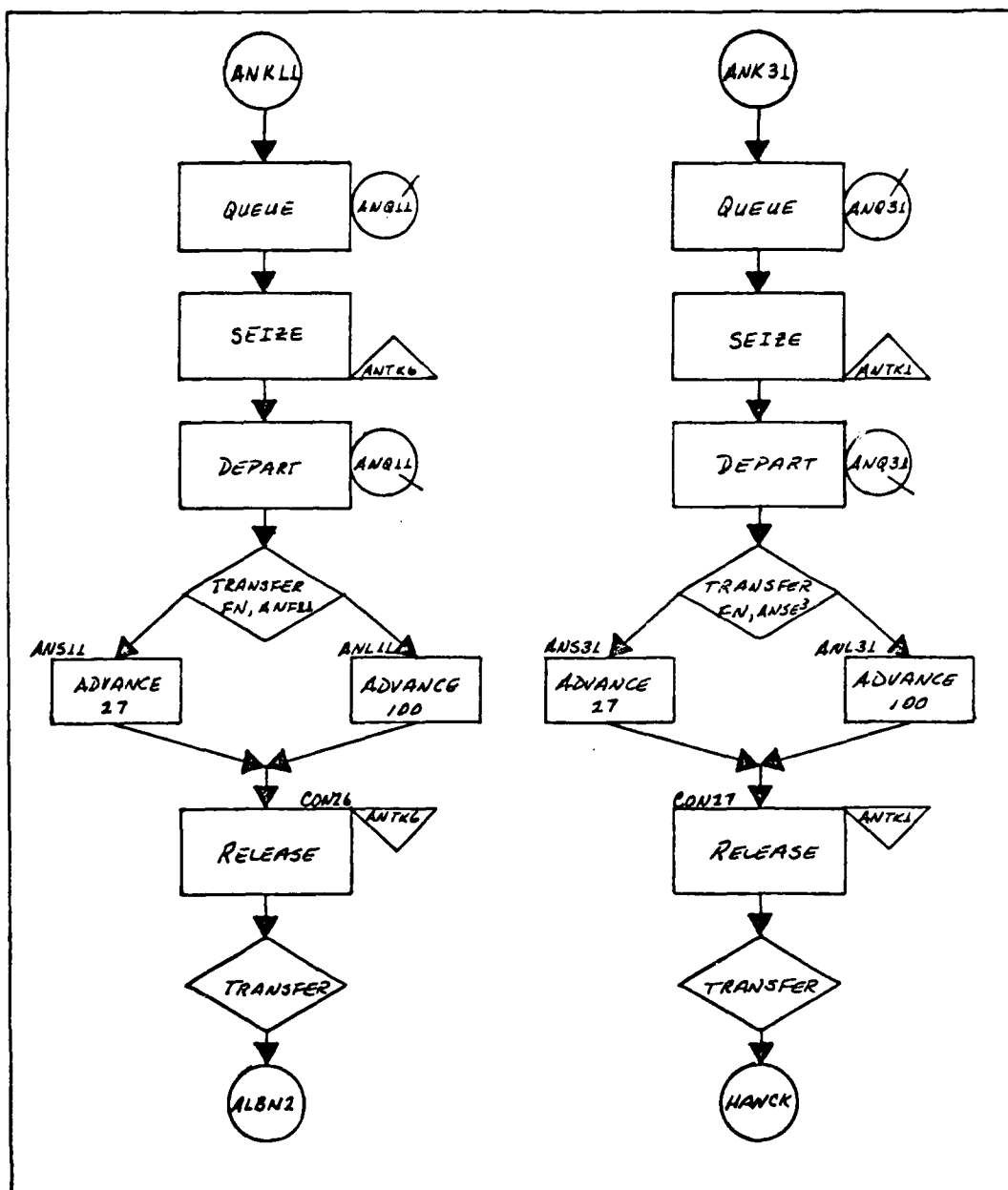


Fig 21. (CONT'D)

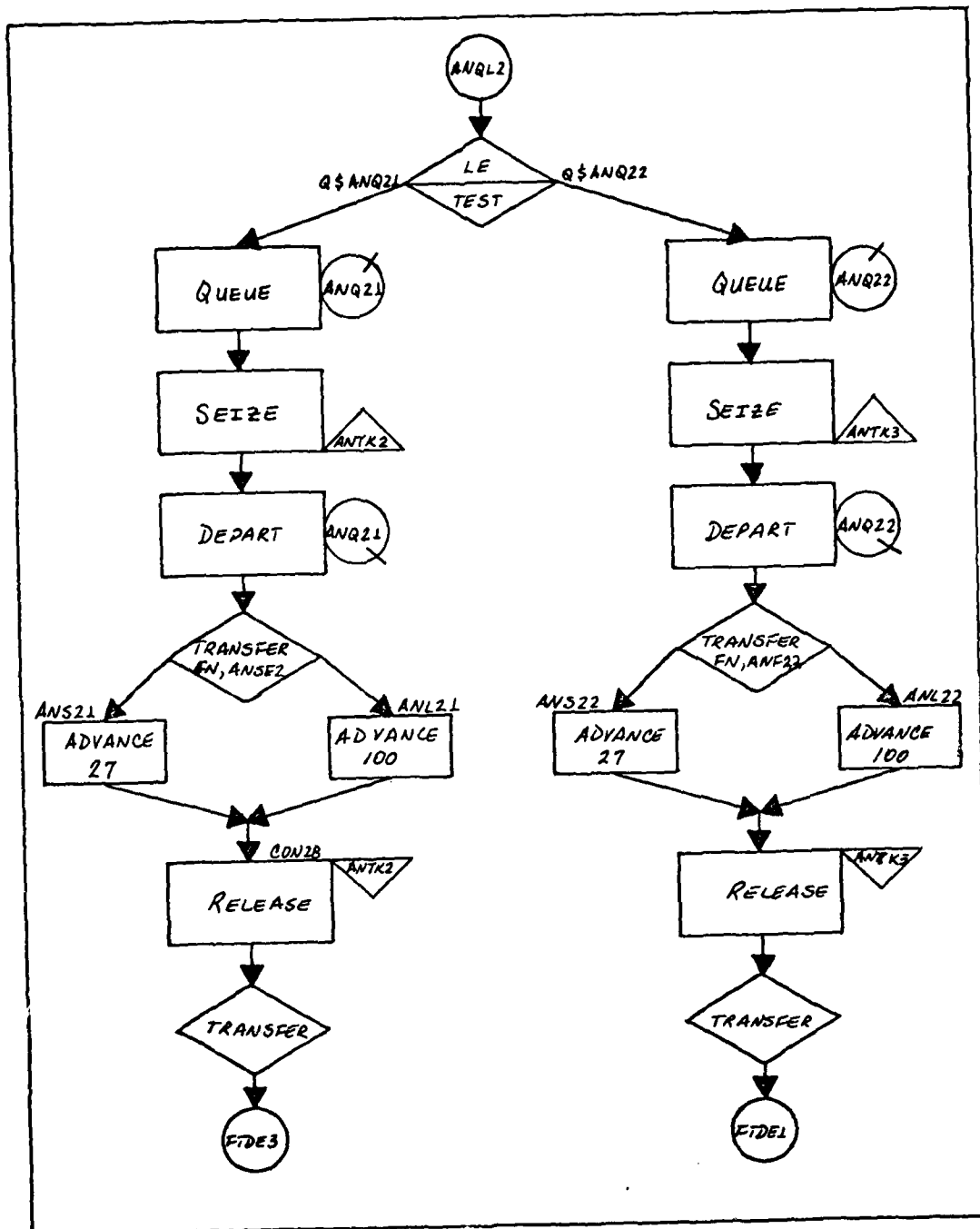


Fig 21. (CONT'D)



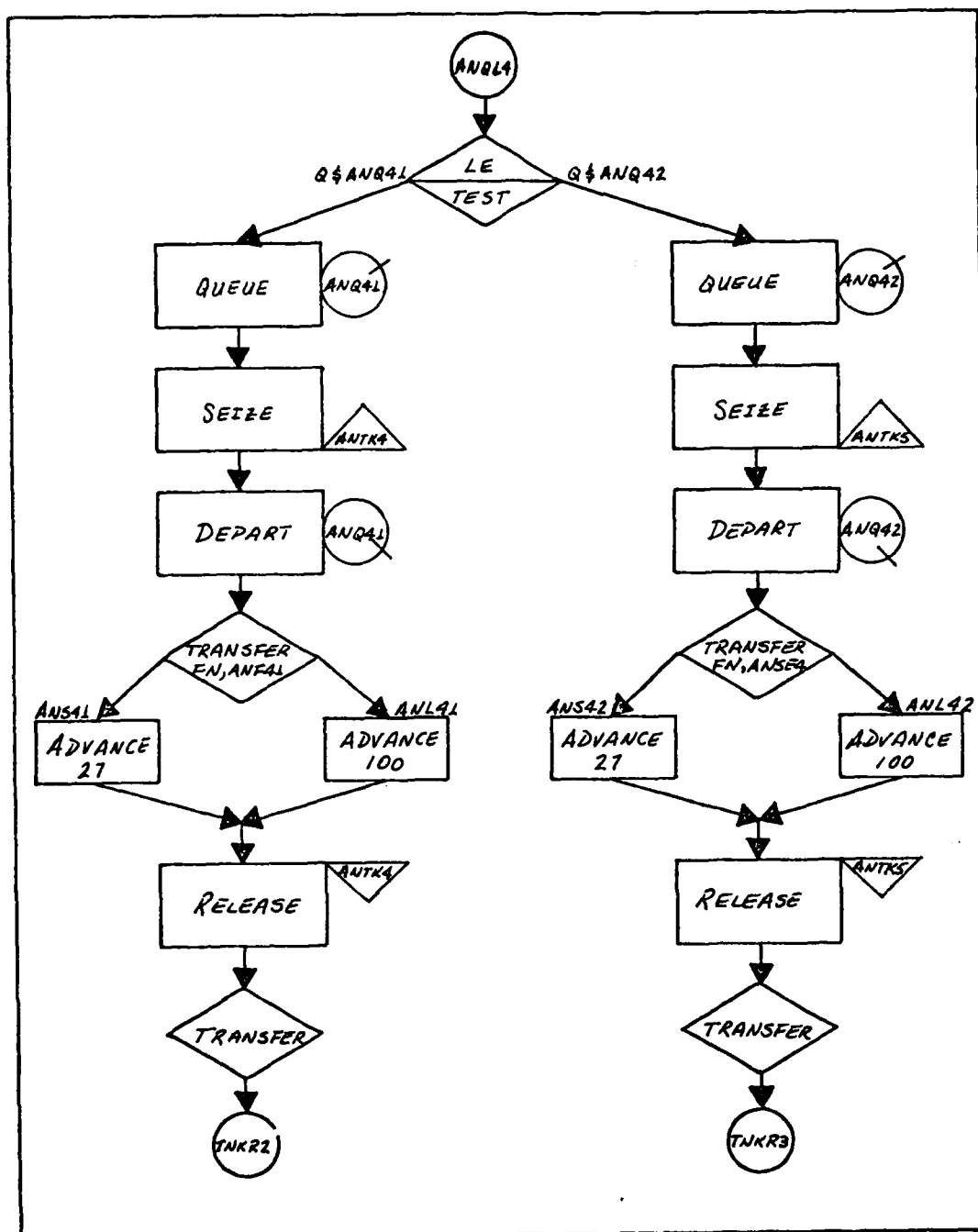


Fig 21. (CONT'D)

block uses the FUNCTION EXPON to determine the exponential distribution of the packets. Random number generator 1, RN1, is used to generate a random number from .00000 to 1.00000. This random number is then used to select which multiplication factor (from the FUNCTION EXPON) to multiply against the given mean interarrival rate (1st operand of GENERATE block) of the packet. The result of the multiplication becomes the number of time units (in this case milliseconds) which must pass before the next packet is generated. The packet priority, packet length, and source node identification number are assigned to the packet's second, third, and fourth parameter values respectively by the ensuing ASSIGN blocks. The packet's destination node identification (ID) number is generated using FUNCTION DEST2. Random number generator two, RN2, is used to generate a random number from .00000 to 1.00000. That number is used to select a number from 1 to 8 which becomes the destination node ID number. It is stored in parameter 1 of the packet.

The packet is transferred to either SCM 1 (ANDY1) or SCM 2 (ANDY2) where it is queued for input processing. For ease of explanation, SCM 1 is assumed to process the packet. When the processor is available, the queued packet seizes it and the packet departs the queue. FUNCTION PRI21 uses the value in parameter 2 of the packet (1 for non-priority and 2 for priority) to determine how much processing time the packet acquires (5 milliseconds for non-priority and 3 milliseconds for priority packets). The packet releases

the processor and moves into the SAVEVALUE blocks. In this simulation program, the SAVEVALUE blocks are used to store the lengths of the output trunk queues. The SAVEVALUES are used later on in the program to assist in determining which trunk to route the packets over. SAVEVALUES 8 through 13 are used for saving the queue lengths at the Andrews node. Everytime a packet is input processed (accomplished by the packet executing this sequence of blocks), new current values of the output queue lengths are stored in the SAVEVALUES. When the packet enters the following TRANSFER block, packet parameter 1 (destination node's ID number) is used in FUNCTION ROUT2 to select the link over which to route the packet. In the case parameter 1's value is a 2, the packet is delivered locally. When a packet terminates locally, the delay time the packet has acquired from time of generation to its delivery at its destination is tabulated (by the TABULATE block) in the system delay table RTIME. Depending on its priority and length, the packet's delay time is further tabulated in either the RTHIS table (priority - short), RTHIL table (priority - long), RTLOS table (non-priority - short) or the RTLOL table (non-priority - long). The packet then enters a TERMINATE block where it is removed from the system. For the example that a packet is to be routed to another node, suppose the packet's destination is Tinker (ID = 8). The packet enters the TEST block labeled ANQL4. Since the Tinker link contains two trunks, the decision has to be made which trunk to select. The TEST block

compares the two queue lengths (ANQ41 and ANQ42) and the packet enters either the queue ANQ41 or the queue ANQ42. Again for ease of explanation, assume the packet enters the queue ANQ42. The selected trunk, ANTK5, is seized when available and the packet departs ANQ42. Based on the packet length value in parameter 3 (1 - short, 2 - long), and the FUNCTION ANSE4, the packet acquires either 27 milliseconds if a short packet or 100 milliseconds if a long packet when it enters the appropriate ADVANCE block. Having acquired the correct transmission time, the packet releases the trunk and transfers to Tinker's SCM 3 where it is further processed and delivered.

The following chapter explains the specifics on how the mean interarrival rates are determined, where the values for processing and transmission come from, source node packet routing matrices, and how the simulation is run for the Baseline model.

## VI Initial Baseline Considerations

In order to have something to compare the algorithm version of the simulation model against, a baseline or control program was developed. The basic processes of packet generation, assigning lengths, priorities and destination node IDs to packet parameters are the same in both programs. Packet input and output processing is the same in both programs as is the termination of packets. The only difference in the two programs is the methodology used in selecting trunks and links over which to route the packets.

The baseline model uses static or fixed routing, that is, given a packet currently at node  $i$  destined for node  $j$ , a predetermined link  $k$  is always used. This route is derived from a given network routing matrix to be explained shortly. The algorithm model, of course, uses the hierarchical routing algorithm to determine the best link and therefore the best trunk over which to route the packet. The next chapter explains the incorporation of the routing algorithm into the simulation model.

This chapter explains how the packet generation rate is calculated for each node. Since each node does not have the same number of Host computers and terminals connected to it, the number of packets entering the node (from the Host computer and terminals) per unit of time is different for each node. The procedure used to calculate the packet mean interarrival rate (time between arrivals

of packets to the node) is explained in detail. The assignment of a destination to a packet is based on the fact that a certain percentage of the packets originating at a node goes to each of the other nodes including the originating, or source node. These percentages and how they are calculated for each node are discussed along with how the percentages are implemented in the baseline program. The ideas of different packet priorities and accumulation of delay time are also discussed in this chapter. The Network Routing Matrix (Ref 17:11), is used to determine which node processes the packet next, given the source node and destination node of the packet. The part it plays in the simulation model is also explained in this chapter. The last section of this chapter explicitly explains how the simulation model was run and gives some sample results of the baseline simulation model.

#### Packet Interarrival Times and Length Determinations

The calculation for the packet interarrival times is a complex process. It involves the use of predetermined statistical data (Ref 17:V-B-110), assumptions made by Systems Control Incorporated (SCI) during their development and partial simulation of the hierarchical routing algorithm (Ref 18:1-2), and a limited number of educated assumptions on the author's part. The author's assumptions were confirmed by telephone conversations with DCA representatives through the Defense Communications Engineering Center (DCEC), Reston, Virginia. When used, each of the assumptions referenced

above is explained.

When a packet is generated, among other parameters, it is assigned one of two lengths. A long packet is 4704 bits in length and a short packet is 600 bits in length. The percentage factors of how many long and short packets are generated are the other results of the packet interarrival time calculations.

In their development and partial simulation of the algorithm, SCI calculated that 81.3% (Ref 18:2), of all the bits in the network are data packet bits. The maximum throughput, or trunking capacity (maximum number of bits able to be supported by the total number of trunks in the network) of the AUTODIN II network is simply the number of trunks multiplied by the transmission speed of the trunks. By the calculation  $28 \times 2 \times 56\text{KBPS}$  (the 2 is because each trunk is two-way), the trunking capacity is 3.136 million bits per second (MBPS). The number of data bits in the network then becomes 81.3% of 3.136 MBPS or 2,549,568 data packet bits. A determination has to be made as to how those bits enter the network, that is, how many of those bits enter the network from Albany, Andrews, and the rest of the nodes.

There are two types of subscribers (users) that connect to nodes and send data into the network. They are Host and terminal subscribers. Host subscribers are large information or computational sources (referred to also as host systems or host computers) (Ref 17:V-1-55). Terminal subscribers, which include terminal computers, are generally mini- to

small-size computers that serve as communications handlers or terminal concentrators and may also provide some limited local computational capacity (Ref 17:V-1-55). They also include input-output devices such as teletypewriters (TTYs), cathode ray tube (CRT) terminals, printers, remote batch terminals, card and magnetic tape terminals (Ref 17:V-1-56). Terminal users input data that is formatted into a combination of long data packets (4704 bits) and short data packets (600 bits) (Ref 18:1). Host users input data that is formatted into long data packets only (Ref 18:1). The percentage of short packets generated from terminal users is 81.25% and the percentage of long packets generated is 18.75% (Ref 18:1).

The actual statistics to which the above percentages apply is in the form of information provided by DCA (Ref 17:V-B-110) as shown in Table II. Since the adjusted trunk capacity figure of 2,549,568 bits is for data packets (including a network overhead of 528 bits of network information), the size of a long data packet is  $4704 + 528$  bits, or 5232 bits. The size of a short data packet thus becomes  $600 + 528$  or 1128 bits. These two numbers are used as shown in Table III, in order to calculate normalized figures. Normalization is necessary because the total number of bits input from Host and terminal subscribers is slightly less than the adjusted trunk capacity of the network. An assumption on the part of the author is that the figures shown in Table III represent the maximum number of bits input into the



TABLE II

## Subscriber Input by Node

(Derived from projected Busy Hour Statistics)

Switch	Hosts		Terminals		Total # of packets input per second
	Send Traffic (BPS)	Equivalent # of long packets	Send Traffic of long (BPS) packets	Equivalent # of short packets	
Albany	168965	35.919	16694	.665	59.191
Andrews	434927	92.459	61750	2.461	178.543
Ft. Detrick	50310	10.695	2667	.106	14.413
Gentile	210813	44.816	33285	1.327	91.218
Hancock	157127	33.403	10564	.421	48.130
McClellan	147544	31.366	20861	.831	60.447
Norton	179069	38.067	9061	.361	50.698
Tinker	498389	105.950	29484	1.175	147.053
	<u>1847144</u>	<u>392.675</u>	<u>184366</u>	<u>7.347</u>	<u>649.693</u>

Total number of bits in network as a result of maximum subscriber input (plus overhead =

392.675 x 5232 = 2054475.6 bits (Hosts)

7.347 x 5232 = 38439.5 bits (Terminals, long packets)

249.671 x 1128 = 281 628.9 bits (Terminals, short packets)

2374544.0 bits Total input

TABLE III

## Normalized Subscriber Input

Number of Data Packet Bits in Network in any given/second Busy Hour period =  
(from trunk capacity figure)

$$3136000 \text{ BPS} \times .813 \times 1 \text{ second} = 2549568 \text{ bits}$$

$$\text{Normalization factor} = \frac{2549568}{2374544} = 1.0737$$

This factor is now multiplied against each of the Host and terminal subscriber inputs to determine the normalized input packet rate at each node.

80

Switch	Hosts		Terminals			
	Send Traffic (BPS)	Equivalent # of long packets	Send Traffic (BPS)	Equivalent # of long packets	Equivalent # of short packets	Total # of packets input per second
Albany	181417	38.566	17924	.714	24.274	63.554
Andrews	466981	99.273	66300	2.642	89.785	191.700
Ft. Detrick	54018	11.483	2863	.114	3.877	15.474
Gentile	226350	48.118	35738	1.424	48.397	97.939
Hancock	168707	35.864	11342	.452	15.360	51.676
McClellan	158417	33.677	22398	.893	30.332	64.902
Norton	192266	40.872	9728	.388	13.175	54.435
Tinker	<u>535120</u> <u>1983276</u>	<u>113.758</u> <u>421.611</u>	<u>31656</u> <u>197949</u>	<u>1.262</u> <u>7.889</u>	<u>42.870</u> <u>268.070</u>	<u>157.890</u> <u>697.570</u>

TABLE III (CONT'D)

Total number of bits in network as a result of maximum subscriber input (plus overhead) =

421.61 x 5232 =	2205863.5 bits	(Hosts)
7.89 x 5232 =	41280.5 bits	(Terminal, long packets)
268.07 x 1128 =	302383.0 bits	(Terminal, short packets)
<u>697.57</u>	<u>2549527.0 bits</u>	

which is .0016% under the adjusted trunk capacity of 2549568 bits (difference is due to round-off errors made during the calculations)

Percent of long packets =  $\frac{429.50}{697.57} = 62\%$

Percent of short packets =  $\frac{268.07}{697.57} = 38\%$

network if the Host and terminal subscribers (currently scheduled to be part of the AUTODIN II network) input data at their respective maximum rates. Theoretically, if this were to occur, the entire network would be in a state of saturation. The Host and terminal subscribers input 81.3% of the trunking capacity of 3.136 MBPS and the nodes themselves add the additional 18.7% due to required network control and synchronization information. The last thing shown in Table III is the determination of the percentages of long and short packets (62% of all packets input are long and 38% are short packets).

Table IV shows the input packet rate per node at the maximum subscriber input rate. Under normal operating conditions, the subscribers would input at about 20% of the rate shown. Thus, the network would nominally operate at 20% trunk loading as confirmed from DCA by telephone conversation. Table IV shows what the input rates, in packets per second (PPS), into the nodes would have to be in order to produce trunk loading of 100% (saturation), 90%, 80%, 70%, 60%, 50%, 40%, and 20% (normal operating traffic level). Also shown in Table IV are the packet interarrival times (inverse of the input packet rate) in milliseconds per packet (MSPP). These constitute the mean packet interarrival times used to generate packets in the simulation model using the GENERATE blocks. Appendix A is the portion of the baseline simulation program representing the code necessary for the Andrews node. The code closely resembles the Block Diagram

TABLE IV

## Packet Input Rates and Interarrival Times

Switch	Saturation Levels							
	100%	90%	80%	70%	60%	50%	40%	20%
Albany								
Input Rate (PPS)	63.55	57.19	50.84	44.48	38.13	31.77	25.42	12.71
Interarrival Times (MSPP)	16	17	20	22	26	31	39	79
Andrews								
Input Rate (PPS)	191.70	172.53	153.36	133.19	155.02	95.85	76.68	38.34
Interarrival Times (MSPP)	5	6	7	8	9	10	13	26
Pft. Detrick								
Input Rate (PPS)	15.47	13.92	12.37	10.42	9.28	7.73	6.19	3.09
Interarrival Times (MSPP)	65	72	81	96	108	129	161	323
Gentile								
Input Rate (PPS)	97.94	88.14	78.35	68.55	58.76	48.97	39.17	19.50
Interarrival Times (MSPP)	10	11	13	15	17	20	26	51
Hancock								
Input Rate (PPS)	51.67	46.50	41.33	36.17	31.00	25.83	20.67	10.33
Interarrival Times (MSPP)	19	22	24	28	32	39	48	97
McClellan								
Input Rate (PPS)	64.90	58.41	51.92	45.43	38.94	32.45	25.96	12.98
Interarrival Times (MSPP)	15	17	19	22	26	31	39	78

AD-A080 484 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCH00--ETC F/0 17/2  
AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II.(U)  
DEC 79 J A WHITTENTON  
UNCLASSIFIED AFIT/SCS/EE/79-15 NL

2 of 5  
AD-A080 484

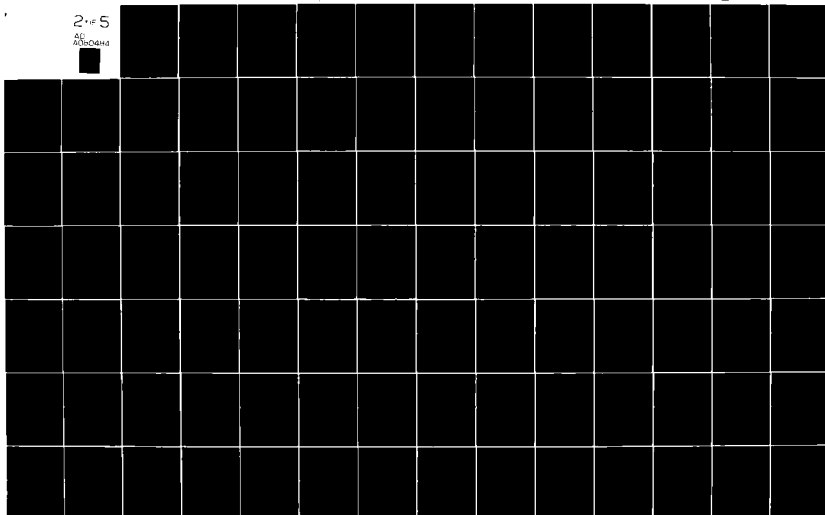


TABLE IV (CONT'D)

<u>Switch</u>	Packet Input Rates and Interarrival Times		<u>Saturation Levels</u>							
			100%	90%	80%	70%	60%	50%	40%	20%
Norton										
Input Rate (PPS)			54.44	48.99	43.55	38.11	32.66	27.22	21.77	10.89
Interarrival Times (MSPP)	18		20	23	26	31	37	46	92	
Tinker										
Input Rate (PPS)			157.89	142.10	126.31	110.52	94.73	78.94	63.15	31.58
Interarrival Times (MSPP)	6		7	8	9	11	13	16	32	

code in Figure 21, Chapter V. The encircled "A" shows the GENERATE block instruction with the corresponding packet mean interarrival time of 26 milliseconds exponentially distributed by the FUNCTION EXPON. Further explanation of the use of the different packet interarrival times is forthcoming in a later section of this chapter.

#### Priority, Length and Destination Assignments

In order for a packet to receive preferential treatment (be processed or transmitted before other packets even though it entered a queue after the other packets) it is flagged or assigned a priority level higher than those of the other packets. In the simulation model a packet is either a priority or a non-priority packet. A priority packet does not preempt or interrupt a non-priority packet that is being processed by an SCM or transmitted on a trunk. Priority packets do move to the head of the queue by passing non-priority packets. Ninety nine percent of the input packets are non-priority, and one percent is given the priority assignment (Ref 18:3). This process is accomplished at the encircled "B" in Appendix A. The TRANSFER instruction sends 1% of the packets to the instruction labeled HIGH2 where they are given the priority status of 2 (stored in parameter 2 of the packet). The other 99% of the packets are given the non-priority status of 1 (stored in parameter 2 of the packet). The values in parameter 2 are used later on to determine the amount of processing time a packet requires.



As determined in Table III, 62% of the packets generated are long packets and 38% are short packets. The simulation model accomplishes this task at the encircled "C" in Appendix A. The TRANSFER instruction labeled CON21 sends 62% of the packets to label LONG2 where parameter 3 of those packets are given the value of 2 denoting a long packet. The other 38% of the packets fall through TRANSFER instruction and are given the value of 1 in parameter 3 denoting a short packet. The values assigned to parameter 3 of each packet are used later on to determine the amount of transmission time each packet requires.

In determining which node is to be the destination of a packet, the information (Ref 17:V-B-131) in Table V is used. It represents node-to-node data packets exchanged for a given time unit. Each row represents the number of packets terminating locally (source node and destination node the same) and the number of packets the source node sends to all the other nodes. From this information Table VI is derived. Instead of the number of packets sent to each of the other nodes, entries in Table VI represent the ratios of packets transmitted to each of the other nodes from a source node. The ratios are calculated, using Table V, by summing up each row to obtain the total number of packets sent by a source node. Each column entry, given the source node, is divided by the appropriate row total to calculate the ratio of packets sent to a particular destination node. At the encircled "D" in Appendix A, parameter 1 of a packet is

TABLE V

## Node-to-Node Packet Exchange

TO:	ALBANY	ANDREWS	FT. DETRICK	GENTILE	HANCOCK	MCCLELLAN	NORTON	TINKER	ROW TOTAL
FROM:									
ALBANY	15.88	24.70	1.72	8.22	7.21	4.76	10.12	38.70	111.31
ANDREWS	35.66	127.11	12.81	20.80	35.28	17.20	17.14	31.86	297.76
FT. DETRICK	1.55	13.53	0.32	2.58	2.07	4.48	0.12	7.10	31.75
GENTILE	13.68	26.31	3.07	43.33	8.63	16.72	10.79	23.81	146.34
HANCOCK	6.41	38.53	3.28	7.05	17.78	5.89	5.80	15.79	100.53
MCCLELLAN	6.56	14.00	4.25	6.00	5.28	33.67	9.19	22.01	100.96
NORTON	16.70	28.22	0.13	7.87	12.55	18.41	13.80	15.11	112.79
TINKER	43.49	41.61	6.92	44.79	27.07	43.15	21.42	88.02	316.47

TABLE VI

Node-to-Node Packet Exchange Ratios										
FROM:		TO:	ALBANY	ANDREWS	FT. DETRICK	GENTILE	HANCOCK	MCCLELLAN	NORTON	TINKER
ALBANY			.1427	.2219	.0155	.0738	.0648	.0428	.0909	.3476
ANDREWS			.1194	.4269	.0430	.0699	.1185	.0578	.0575	.1070
FT. DETRICK			.0488	.4261	.0101	.0813	.0652	.1411	.0038	.2236
GENTILE			.0935	.1798	.0210	.2961	.0590	.1143	.0737	.1626
HANCOCK			.0638	.3833	.0326	.0701	.1769	.0586	.0577	.1570
MCCLELLAN			.0650	.1387	.0421	.0594	.0523	.3335	.0910	.2180
NORTON			.1481	.2502	.0012	.0697	.1113	.1631	.1224	.1340
TINKER			.1374	.1315	.0219	.1415	.0856	.1363	.0677	.2781

assigned a destination node ID number using the FUNCTION DEST2. The function, encircled "E" in Appendix A, uses random number generator two, RN2, to generate a random number from .00000 to 1.00000. The next line following DEST2 contains eight discrete intervals, each interval separated by a slash. The decimal numbers in the intervals are the cumulative sums of Andrew's row in Table VI. The difference between the  $n^{\text{th}}$  and  $n^{\text{th}} + 1$  intervals are the ratio entries in Andrew's row of Table VI. For example, the difference between the 4th and 5th entries is .7777 - .6592, or .1185. Looking at Table VI, this number corresponds to the ratio of packets originating at Andrews with Hancock as the destination node. After the .7777 entry is the number 5, which is Hancock's node ID number. If the generated random number is between .6592 and .7777, then parameter 1 of the packet currently being processed is assigned the integer number 5, denoting the destination node of the packet to be Hancock. The rest of the intervals in DEST2 are derived in the same fashion, that is, for random number values in the range .00000 to .11940, the packet is assigned destination node ID of 1 (Albany), .11941 to .54630 (.1194 + .4269) the ID is 2 (Andrews), .54631 to .5893 (.5463 + .0430) the ID is 3 (Ft. Detrick), and so on.

All of the pertinent packet parameter values have now been assigned to the packet. Parameter 1 contains the destination ID number (1 to 8), parameter 2 contains the priority status (1 - non-priority, 2 - priority), parameter 3 con-

tains the packet length (1 - short, 2 - long), and parameter 4 contains the number 2 which indicates Andrews as the packet's source node. The packet is now ready to enter one of the two SCMs to be processed and transmitted on to the next node (which might be its destination node or a tandem node). This is accomplished by the TRANSFER instruction (encircled "F" in Appendix A).

#### Packet Processing and Transmission

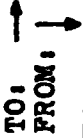
Not only do new packets just generated enter either ANDY1 or ANDY2, but also the packets sent to the Andrews node from other nodes enter Andrews at these two entry points. For ease of explanation, assume the packet is sent to SCM 1. In order to receive service, the packet enters the processor's input queue, seizes the processor when it is available, and departs the processor's input queue as shown in Appendix A, encircled "G". If the packet is a priority packet, it moves to the front of the FIRST-IN-FIRST-OUT (FIFO) input queue ANQU1. If the processor ANPS1 is busy (currently seized by another packet), the queued packet remains in the queue until the processor is free. When the packet seizes the processor it acquires a certain amount of processing delay time depending on its priority (encircled "H", Appendix A). To determine the priority of a packet, FUNCTION PRI21 (encircled "I") is used by the TRANSFER instruction. Packet parameter 2 is inspected and if it is a 1 (non-priority), the packet is transferred to the label LOP21 where the ADVANCE instruction

adds 5 milliseconds (Ref 17:V-B-26) to the packet's system time parameter (located in GPSS's system tables). If the packet parameter is a 2 (priority), the packet transfers to the label HIP21 where the ADVANCE instruction adds 3 milliseconds (Ref 17:V-B-26) to the packet's system time parameter. After receiving the appropriate amount of SCM processing time, the processor is released (encircled "J", Appendix A).

The packet now enters the SAVEVALUE instructions (encircled "K", Appendix A). ANQ11 through ANQ42 are the output queues associated with the output trunks emanating from the Andrews node. The lengths of the queues, or number of packets in each queue, are stored in SAVEVALUES 8 through 13. All the necessary conditions for the packet to start the transmission process have now been accomplished.

When the packet enters the TRANSFER instruction located at the encircled "L", Appendix A, the FUNCTION ROUT2 (encircled "M", Appendix A) uses the value in packet parameter 1 to determine which outgoing link on which to queue the packet. When the destination node of a packet is not a directly connected node to the node processing the packet, a tandem or intermediate node is selected. Due to the 2-hop constraint, if a node is a tandem node it is directly connected to the destination node. Table VII (Ref 18:11) is used in the next-node determination. Given the source node (row headings), and the destination node (column headings), the entries give the next node to process the packet (inc-

TABLE VII  
Network Base Routing Matrix

TO: FROM: 	ALBANY	ANDREWS	FT DETRICK	GENTILE	HANCOCK	MCCLELLAN	NORTON	TINKER
ALBANY		ANDREWS	FT DETRICK	FT DETRICK	ANDREWS	NORTON	NORTON	TINKER
ANDREWS	ALBANY		FT DETRICK	FT DETRICK	HANCOCK	TINKER	ALBANY	TINKER
FT DETRICK	ALBANY	ANDREWS		GENTILE	HANCOCK	GENTILE	ALBANY	TINKER
GENTILE	FT DETRICK	FT DETRICK	FT DETRICK	GENTILE	HANCOCK	MCCLELLAN	NORTON	TINKER
HANCOCK	ANDREWS	ANDREWS	FT DETRICK	GENTILE	HANCOCK	GENTILE	GENTILE	ANDREWS
MCCLELLAN	NORTON	TINKER	GENTILE	GENTILE	GENTILE	MCCLELLAN	NORTON	TINKER
NORTON	ALBANY	ALBANY	ALBANY	GENTILE	GENTILE	MCCLELLAN	NORTON	GENTILE
TINKER	ALBANY	ANDREWS	FT DETRICK	GENTILE	ANDREWS	MCCLELLAN	GENTILE	TINKER

cluding tandem and destination nodes). The values of the FUNCTION ROUT2 correspond to Andrews' row of Table VII. The link labels in ROUT2 are the links to route packets on to get to the node given in Andrews' row. For example, packets destined for Gentile (ID of 4) from Andrews pass through Ft. Detrick. Where Andrews' row and Gentile's column intersect (in Table VII), Ft. Detrick is the node given to process the packet next. For Gentile being the packet's destination, parameter 1's value is a 4. The 4th entry in ROUT2, each entry separated by a slash, gives the label ANQL2 which is the link connecting Andrews and Ft. Detrick. If the value is a 1, the packet transfers to label ANK11, or if it is a 2, the packet transfers to the label TERM2 for local "delivery" (termination). As seen in the rest of the function, for each destination ID number (1 to 8), a label exists where the packet is transferred to. In the event a link contains more than one trunk, as in the link between Andrews and Ft. Detrick, ANQL2 (encircled "N", Appendix A), the individual trunk queue lengths are compared to queue the packet on the shorter of the two. If the packet is to be sent to Ft. Detrick, the output queue lengths (ANQ21 and ANQ22) are compared. If the length of ANQ21 is shorter than ANQ22, the packet is sent to label ANK21 (primary trunk to Ft. Detrick) otherwise it is sent to label ANK22 (secondary trunk to Ft. Detrick). Since the link to Tinker (ANQL4) contains two trunks, a similar process is used for trunk selection. For packets going to Albany or Hancock, they are sent directly



to the respective trunks (ANK11 or ANK31).

Locally terminating packets are sent to the label TERM2 (encircled "P", Appendix A) where the TABULATE instruction enters the packet's accumulated delay time (value in the packet's system time Parameter M1) in a table labeled RTIME (defined encircled "Q", Appendix A). The Table RTIME defines the system parameter to be tabulated (M1), the first boundary point (20), width of each intermediate table interval (20), and the total number of intervals in the table (200). For example, if a packet had accumulated 15 milliseconds of delay time, it would be tabulated in the first interval 0-20. If another packet has amassed a total of 93 milliseconds (msec) of delay time, it would be tabulated in the table's interval 80 - 100. Table VIII is an example of a sample output of system table RTIME. The UPPER LIMIT column shows the upper limit of the intervals, that is, an upper limit of 20 represents the interval 0 - 20 msec, 60 represents the interval 41 - 60 and so on. The OBSERVED FREQUENCY column gives the total number of packets whose delay times are within the corresponding time interval. The other entries are self-explanatory. After the packet's delay time is tabulated in RTIME, the program tabulates the packet's delay time in other tables depending on the packet's priority and length as shown in the section of code at the encircled "R", Appendix A. The packet finally leaves the network by entering one of the TERMINATE instructions.

For non-priority packets, assume the packet's parameter

TABLE VIII

## Example Output of System Delay Table RTIME

TABLE RTIME						
Entries in Table		Mean Argument	Standard Deviation		Sum of Arguments	
355		104.045	110.028		36936.000	
TABLE RTIME						
Upper Limit	Observed Frequency	Per Cent of Total	Cumulative Percentage	Cumulative Remainder	Multiple of Mean	Deviation From Mean
20	131	36.90	36.90	63.10	0.192	-0.764
40	27	7.61	44.51	55.49	0.384	-0.582
60	6	1.69	46.20	53.80	0.577	-0.400
80	12	3.38	49.58	50.42	0.769	-0.219
100	4	1.13	50.70	49.30	0.961	-0.037
120	54	15.21	65.92	34.08	1.153	0.145
140	21	5.92	71.83	28.17	1.346	0.327
160	12	3.38	75.21	24.79	1.538	0.509
180	7	1.97	77.18	22.82	1.730	0.690
200	11	3.10	80.28	19.72	1.922	0.872
220	22	6.20	86.48	13.52	2.114	1.054
240	10	2.82	89.30	10.70	2.307	1.236
260	6	1.69	90.99	9.01	2.499	1.417
280	4	1.13	92.11	7.89	2.691	1.599
300	5	1.41	93.52	6.48	2.883	1.781
320	7	1.97	95.49	4.51	3.075	1.963
340	4	1.13	96.62	3.38	3.268	2.144
360	2	0.56	97.18	2.82	3.460	2.326
380	3	0.85	98.03	1.97	3.652	2.508
400	1	0.28	98.31	1.69	3.844	2.690
420	2	0.56	98.87	1.13	4.037	2.872
440	0	0.00	98.87	1.13	4.229	3.053
460	0	0.00	98.87	1.13	4.421	3.325
480	1	0.28	99.15	0.85	4.613	3.417

TABLE VIII (CONT'D)

TABLE RTIME	Upper Limit	Observed Frequency	Per Cent of Total	Cumulative Percentage	Cumulative Remainder	Multiple of Mean	Deviation From Mean
	500	0	0.00	99.15	0.85	4.806	3.599
	520	1	0.28	99.44	0.56	4.998	3.780
	540	0	0.00	99.44	0.56	5.190	3.962
	560	1	0.28	99.72	0.28	5.382	4.144
	580	0	0.00	99.72	0.28	5.575	4.326
	600	0	0.00	99.72	0.28	5.767	4.508
	620	0	0.00	99.72	0.28	5.959	4.689
	640	1	0.28	100.00	0.00	6.151	4.871

All remaining classes are zero.

1 contains the value of 8 (Tinker's ID). Assume also after comparing trunk queue lengths, the packet transfers to ANK42 (encircled "S", Appendix A). The packet enters queue ANQ42 which corresponds to the 4th link, 2nd trunk queue. When the trunk ANTK5 is available, the packet seizes it and departs ANQ42. Depending on the packet's parameter 2 value, the packet is transferred to ANS42, if a short packet, to acquire 27 msec transmission time, or to ANL42, if a long packet, to acquire 100 msec. The transmission times of 27 and 100 msec are derived by dividing the packet's length by the transmission speed of the trunk (56000 KBPS) plus a propagation factor of 7 msec. After acquiring the appropriate transmission time, the packet releases ANTK5 and transfers to Tinker's SCM 3, TNKR3, for further processing.

#### Running the Baseline Model

In any simulation, the first question to ask is "What exactly do I want the simulation model to show?" In the baseline simulation, and in the algorithm simulation, there are several items of interest such as mean packet delay, maximum packet delay, and adaptability (reaction characteristics) of the routing discipline to topological changes and traffic level fluctuations. The purpose of this section is to explain exactly how the baseline simulation model is run on the CDC CYBER 175 computer in order to show the items of interest detailed above. Due to time and resource constraints

only the traffic level is allowed to fluctuate. In the next chapter, the incorporation of the hierarchical routing algorithm is explained and both topological changes and traffic level fluctuations are simulated. The same procedure used in the baseline program to vary the traffic levels holds for the algorithm simulation model also.

Basically, the model is run for five seconds at 20% saturation in order to establish a steady-state system. The program is then allowed to run for thirty seconds, again at a 20% saturation. Thirty-five seconds into the simulation, a traffic volume spike occurs for one second, then the traffic volume returns to the normal operating level of 20% of saturation and remains there for twenty more seconds. Four runs are made with the spiked traffic level being 60%, 70%, 80%, and 90% of network saturation. For each of the four runs statistics are output after thirty second continuous level of 20% saturation, after the one second burst, and once each second thereafter for the next twenty seconds. The reasoning behind so many statistic dumps is to show how the routing discipline (static in the baseline model, adaptive routing in the algorithm model) behaves after the one second traffic burst.

The operating system of the CYBER 175 allows remote terminals (CRTs and Texas Instrument's Silent-700 terminals) to access the main computer through a portion of the operating system called Intercom. Using a normal LOGIN procedure, the interactive user can create, manipulate, and save files

(usually programs) once into the Intercom portion of the operating system. The GPSS code for the baseline program was entered similarly and saved on permanent file. To make a set of computer runs, a copy of the baseline program is called into a file editor and made a local file (local as long as the Intercom session lasts) with the file name of say, DIN.

For a set of four runs, the only section of code altered is the section denoted at the encircled "T", Appendix A.

GPSS allows redefinition, or overlaying, of previously labeled instructions with new instructions labeled with the same label name. For the first run, (spiked traffic at 60% saturation), the code denoted by the encircled "U" is deleted from the edit buffer. The rest of the edit buffer which not only contains the code as shown in Appendix A for the Andrews node but also similar code for each of the other seven nodes in the network, is saved under another local file name (say, RUN1). Appropriate Intercom control commands are entered through the keyboard to start the compilation and execution of the program with the file name of RUN1. The file with the name DIN is then called back into the edit buffer to be modified for the second run. This time the section of code denoted by the encircled "V" is deleted from the edit buffer. The rest of the edit buffer is saved under a new local file name (say, RUN2). Again appropriate Intercom control commands are entered to start the compilation and execution of the program with the file name of RUN2. Similarly,

for the 3rd and 4th runs DIN is called back into the edit buffer and changed appropriately following the same procedure as outlined above but removing only those sections of code not relating to spiked traffic level of interest.

During execution, the time-controlling portion of code is denoted by the encircled "W", Appendix A, and each of the remaining START instructions. Each time unit in the simulation program is considered to be one millisecond (msec). Thus the GENERATE 1000 instruction generates one transaction per 1000 msec or one second. The first operand of a START instruction is stored by the GPSS simulation control processor and acts like a time-control counter. After a second of simulation time, the GENERATE 1000 instruction generates a transaction which immediately enters the TERMINATE 1 instruction. The "1" of the TERMINATE instruction subtracts one from the time-control counter. If the counter's value is zero that portion of the simulation is ended. If it is not zero, the simulation runs for another second, the GENERATE 1000 instruction generates another transaction which immediately enters the TERMINATE 1 instruction. One is subtracted from the time-control counter. If the counter is zero, the simulation ends, if not zero, the simulation runs for another second. When the counter reaches zero, the GPSS simulation processor scans the following instructions until it finds another START instruction or the END instruction. If the simulation processor finds labeled instructions between START instructions, the processor overlays the existing

instructions with the same labels in the main portion of the program with the new instructions. This is how the traffic intensity levels are changed in the simulation. The different mean interarrival times in the labeled GENERATE instructions come from Table IV. When the END instruction is encountered, the simulation is ended.

In words, the execution of the program (for the 60% spike) is as follows: At the start of the execution, the simulation clock is set to zero. Execution begins and packets are generated in the main body of the program moving through the network code as instructed. Also at time 0, the GPSS control processor scans down to the first START instruction (START 5, NP) where the NP stands for no statistics print-out after this section of the simulation ends. The time-control counter is set to 5 and execution actually starts. After one second, the GENERATE 1000 instruction (encircled "W"), generates a transaction which immediately enters the TERMINATE 1 instruction. One is subtracted from the time-control counter. If non-zero, simulation continues, if zero, statistics are output and the GPSS processor scans down to the next START instruction (START 30,,30). The RESET instruction clears all the statistics accumulated except for those involving facilities (SCM processors and the trunks) and queues (SCM processor input queues and the trunk queues). The first "30" of the START instruction sets the time-control counter to 30. The second "30" informs the GPSS processor to generate output statistics after 30 transaction terminations



(30 seconds). After 30 more seconds of simulation time, the statistics requested are output and the GPSS control processor scans down to the next START card (START 01,,01). The instructions labeled ALBAN through TINKE in the selection of code denoted by the encircled "X", Appendix A, replaces the instructions under the same label name in the main program. The packet generating instruction in each of the node's code is modified to generate packets at 60% saturation rate. The RESET instruction clears all the statistics except for those given in the instruction. The modified program runs for one second, terminates and statistics are output. The GPSS control processor scans down to the next START instruction (START 1,,1). The instructions denoted by the encircled "Y" are used to reestablish the original packet generation intensity of 20% saturation. The statistics are reset and execution begins again. This procedure occurs again and again until the END instruction is encountered and the complete simulation ends. It should be noted that when a time interval elapses, the status of the model, that is, the number of packets in the queues and packet positions throughout the model does not change. The RESET instruction clears only the appropriate tables (statistics).

### Baseline Results

The results of the baseline simulation are of no real importance except to get a feel of how the basic model operates. Items of interest are the mean packet delay, maximum packet delay and (to see) how these values change when

the traffic intensity increases. In the next chapter these results are compared against similar algorithm model runs to get a feel of the algorithm operation.

The best way to show the results is in the form of graphs. Specific data relative to the above mentioned items of interest are extracted manually from the simulation statistic outputs and put together to form five illustrations, three graphs and two tables. The statistical output data starts at the 30 second time interval. The mean delay value of the 30 second time mark (in parenthesis on the graphs) is the mean packet delay over a 30 second simulation time period during which the system has reached equilibrium. Figure 22 shows the graphical results of data taken from the system delay table, RTIME, for each of the four simulation runs. Also shown in Figure 22 are the significant maximum packet delay times encountered for each run (in parentheses on the graph). These values indicate that during the time interval in which the statistics were taken, a packet terminated having the delay time indicated in parentheses. Figure 23 shows the results of data taken from the non-priority - short packet delay table, RTLOS, along with the significant maximum delay times. Figure 24 shows the results of data taken from the non-priority - long packet delay table, RTL0L, and the associated significant delay times. Since only 1% of the packets generated as priority packets, the data on the priority short and long packets are not completely representative. The number of priority packets generated in the

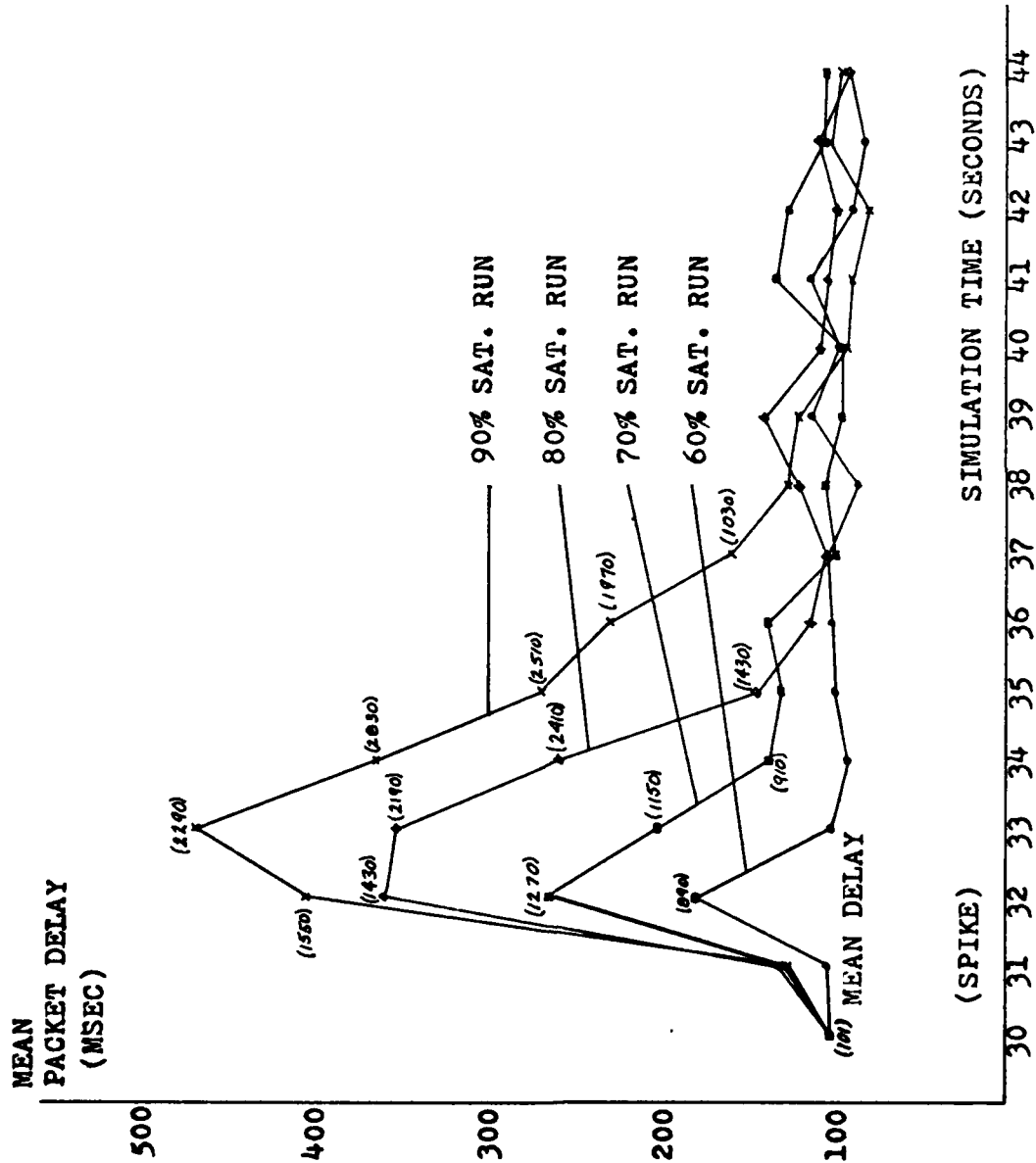


Fig 22. System Delay (all packets)

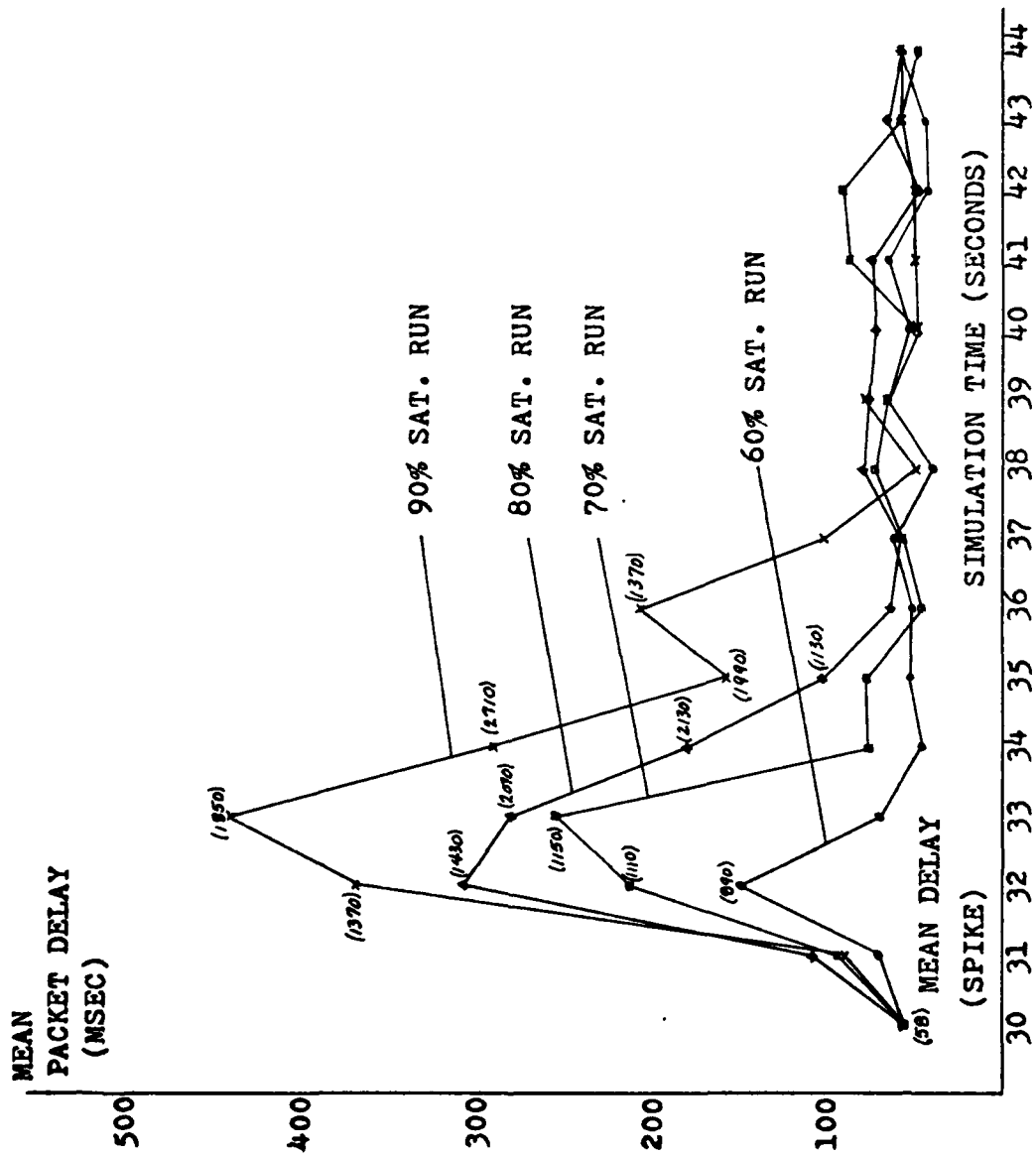


Fig 23. System Delay (non-priority - short packets)

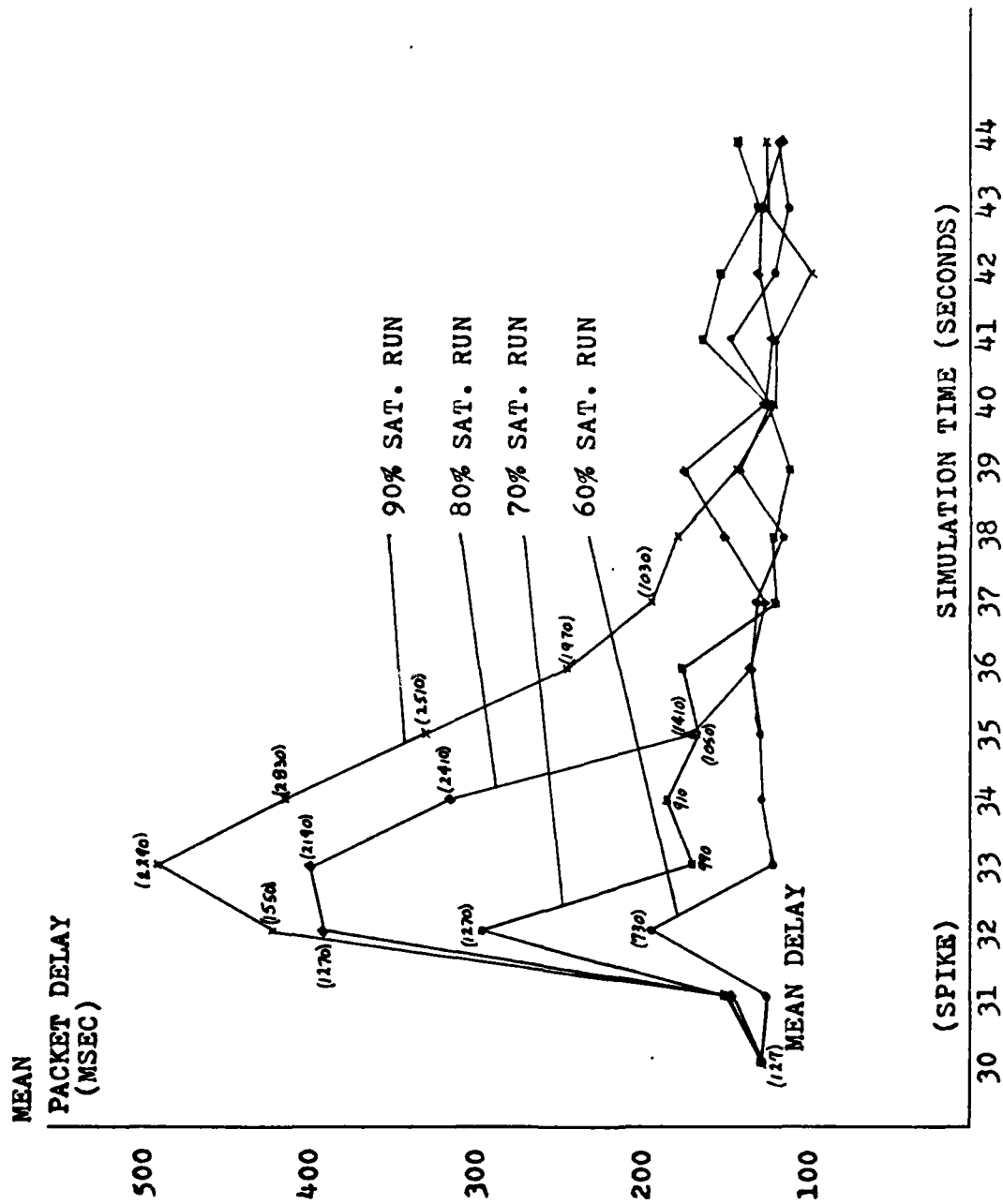


Fig 24. System Delay (non-priority - long packets)

one second time intervals is so small as to infer incorrect mean delay times if shown in graphical form. During some of the one second time intervals, only one priority - short packet is generated and in the other one second time intervals only one priority - long packet is generated. In some of the one second time intervals, no priority packets were generated at all! The best way to show these types of results is in tabular format. Table IX contains data extracted from the statistical outputs for priority - short packets, and Table X shows the data extracted for priority - long packets.

Figure 25 contains information on packet delays extracted from DCA's System Performance Specifications (Ref 19) for the AUTODIN II network.

Category Type	Category I (Priority)		Category I II III IV (Non-priority)	
	Mean (msec)	Max (msec)	Mean (msec)	Max (msec)
Short Packets (1120 bits)	180	580	380	1350
Long Packets (5224 bits)	330	730	530	1500

Fig 25. AUTODIN II Packet Delay Specifications

The delays shown are for the two-hop case, that is, packets are assumed to traverse a source, tandem, and destination node, and SCM processors are operating at eighty per cent utilization (Ref 19:43). The maximum is defined to be the 99th percentile of delivery delay distribution (Ref 19:43), that is, one tenth of one per cent of the packets

TABLE IX

## System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)					
	60% Sat.		70% Sat.		80% Sat.	
	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	49	110	49	110	49	110
31 (spike time)	122	390	8	8	112	130
32	-	-	3	3	-	-
33	154	154	33	33	119	190
34	6	6	-	-	-	-
35	39	39	-	-	-	30
36	-	-	-	-	-	-
37	33	33	-	-	63	63
38	-	-	-	-	36	36
39	-	-	-	-	-	-
40	-	-	33	33	3	3
41	-	-	-	-	-	-
42	-	-	66	130	-	-
43	18	30	-	-	48	50
44	-	-	18	30	18	30

TABLE X

System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)					
	60% Sat.		70% Sat.		80% Sat.	
	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	130	310	130	310	130	310
31 (spike time)	236	390	109	210	154	210
32	258	310	-	-	155	310
33	3	3	-	-	-	-
34	3	3	114	130	162	210
35	150	230	-	-	3	3
36	3	3	108	110	-	-
37	-	-	160	160	106	210
38	124	124	158	210	-	-
39	89	170	209	209	-	-
40	-	-	50	210	204	210
41	55	110	106	110	-	-
42	108	108	107	210	3	3
43	161	161	-	-	176	210
44	3	3	-	-	-	-
					193	193
					3	3
					106	106
					72	210
					106	106
					-	-
					127	150
					73	110
					209	209
					106	106
					124	250
					106	106
					167	230
					209	270
					159	170
					130	310
					90% Sat. Mean	Run Max



in the system are allowed to incur a delay not to exceed the maximum values given in Figure 25.

In the steady-state condition (the first 30 seconds of the simulation), the delay times accorded packets are well within the specifications given in Figure 25. However, when traffic intensity spikes occur, specifically the 80% and 90% spikes, both mean and maximum delay values are exceeded (in the time frames immediately after the spike). The DCA Performance Specifications do not list any concrete behavioral characteristics the routing algorithm should provide other than it should recognize network degradations and react efficiently to compensate for traffic volume fluctuations. As seen in the graphs, the baseline static routing discipline smoothes back out to an average delay value of about 100 milliseconds within six seconds after the spike occurs. This six second time span may or may not be acceptable considering this simulation is only concerned with a control simulation model.

These results, performance characteristics and the values given in Figure 25 take on a more significant meaning in the next chapter. The whole purpose of this paper is shown in the next chapter which explains how the hierarchical routing algorithm is incorporated into the baseline simulation model. The algorithm simulation program illustrates the adaptability of the algorithm by varying the traffic intensities while topological changes occur.

## VII Algorithm Description

As stated in the previous chapter, the basic model and the simulation code of the baseline program remains the same for the algorithm version of the program. The differences are the addition of Fortran subroutines representing the routing algorithm and the addition of GPSS instructions necessary to activate the Fortran subroutines as well as the GPSS code necessary to pass numerical data to the subroutines. Since the algorithm "Background Functions" require periodic entry into some of the subroutines, additional simulation code is added to facilitate these periodic entries.

This chapter explains how the Fortran versions of the algorithm's modules are written and how they merge with the baseline program. The GPSS changes necessary to accomplish and support the new program are also explained. In addition, the chapter explains exactly how the new program is run with respect to normal network configuration, that is, all nodes, links, and trunks operational. The final section of this program explains how pre-selected trunks, links, and nodes are removed from the network and how the algorithm compensates for the degradation.

### Fortran Subroutines for Modules

Each module of the algorithm, MINHOP, RTABLE, PAKROUTE, MESGEN, and UPDATE, is coded in Fortran with each node of the network maintaining its own copy of the algorithm. Each

module has its own Fortran subroutine counterpart except MINHOP. There are four subroutines composing MINHOP: INITIALIZE (same as CONNECTIVITY described in Chapter IV), INITIALIZE\_D, COMPUT\_D, and CONN\_CHANGE. Appendix B represents Ft. Detrick's version of the algorithm. The module names and their corresponding subroutine names are the following:

- (1) MINHOP
  - INITIALIZE - DEINIT
  - INITIALIZE\_D - DEINID
  - COMPUTE\_D - DECOMD
  - CONN\_CHANGE - DECCHG
- (2) RTABLE - DERTBL
- (3) PAKROUTE - DEPKRT
- (4) MESGEN - DEMSGN
- (5) UPDATE - DEUPDT

The COMMON/DETABL/ statement, and the following INTEGER and REAL statements denoted by the encircled "A" in Appendix B contain the algorithm database. The block is in common with each subroutine since each subroutine uses and modifies the algorithm's tables, arrays, variables and constants contained in the block. The names used are as close as possible to the names used in the module flow-charts in Chapter IV. Each name is prefixed with "DE" to denote Ft. Detrick. Each of the other nodes has almost the identical code written for them. What differentiates

one node's code from the other is the first two letters in each name. Albany's code is prefixed with the letters "AL", Andrews "AN", Gentile "GE", Hancock "HA", McClellan "MC", Norton "NO", and Tinker "TI". Other differences are the arrays XXLINS and XXBFID. The values of the XXLINS array (DELINS for Ft. Detrick - encircled "B" of Appendix B), are the numbers of trunks within a particular link indexed by that link. For example, link number 1 contains 3 trunks and link number 4 contains 1 trunk. Since each node does not have the same number of links or trunks, the array is different for each node. The values of the other array, XXBFID (DEBFID for Ft. Detrick - encircled "C" of Appendix B), are the trunk numbers of the first trunk in a particular link indexed by that link. For example, the first trunk in link 2 is trunk number 4 and the first trunk in link number 4 is trunk number 9. The last two differences are the local node's ID number (DESELF = 3 for Ft. Detrick - encircled "D" of Appendix B), and the node saturation threshold XXT2 (DET2 for Ft. Detrick - encircled "E" of Appendix B). Each node does not have the same number of SCM processors. Hence the number of packets on input queue for a one-processor putting it into saturation would not put a three-processor node into saturation since its input queue can, in effect, hold three times the number of packets. Thus the saturation threshold for one-processor nodes is lower than that of the three-processor node. For single processor nodes the threshold is 12, for dual-processor

nodes it is 18, and for triple-processor nodes the threshold is 20. For a more detailed definition of the names used in the subroutines see Appendix C. The names in Appendix C are for the Ft. Detrick node but can be extended to the other nodes by adding the appropriate prefix.

The arguments of the subroutine call and their declarations (encircled "F" of Appendix B) are required, in the order shown, by GPSS. In the main GPSS simulation program the packet's source node ID, destination node ID, priority status and length are stored in parameters associated with that packet. Also stored in the main program (in the SAVEVALUES), are the output trunk queue lengths. The values are passed to the Fortran subroutine via the arguments of the subroutine call. The packet's parameters are passed through the named PARMVA and the queue lengths are passed through the name SAVVXX (SAVVDE for Ft. Detrick). More explanation of the passed values is given in the next section of this chapter.

After the 64 (8 nodes of 8 subroutines each) subroutines are written, they are compiled as one large program and a single relocatable binary file produced. The filename is an input parameter for the command to execute the GPSS simulation program. Before execution starts, the binary file is loaded into the core and the subroutine names compared against all the subroutine calls in the main simulation.

## Simulation Program Modifications

GPSS HELP Blocks. The instructions GPSS provides to communicate with Fortran subroutines are called HELP instructions. Their format is HELPX A, B, C, D, E, F, G, where X is one of A, B, or C, and A through G are the operands. Operand A is the name of the subroutine and B through G are the values passed. HELPA instructions provide one-way communication to Fortran subroutines, that is, the values passed to the subroutine can only be read. HELPB instructions provide for two-way communications. The values passed are the contents of the SAVEVALUES explicitly given in the operands B through G which can be read or changed in the subroutine. HELPC is a very special instruction. Not only does it provide two-way communication but it also makes all the GPSS internal system tables and arrays plus all the attributes concerning queues, facilities, SAVEVALUES, and user defined TABLES available for use and modification by the Fortran subroutine. The B-G operands are read-only but the system values can be read and/or modified. The system values are not passed as arguments in the subroutine call but rather made accessible to the subroutine (see encircled "F" in Appendix B). The order of definition in the subroutine must be the same as shown in the encircled "E" portion because this order is the way the internal tables, arrays, etc. are laid out in core. Any scrambling of the order results in chaos if the values are referenced then modified. There is no control program checking to ensure proper referencing and

usage of the system values.

Algorithm Activation. Appendix D is the complete GPSS simulation program which incorporates the hierarchical routing algorithm. There are essentially seven sections to the program. Sections 1 and 7 make up the original baseline program with minor changes and is called the main program. Sections 2 and 3 provide the necessary periodic entry in the MESGEN subroutine at each of the nodes. Section 2 handles the entry into MESGEN on the FTICK (to check "Bad News") interval, every 100 milliseconds, and Section 3 handles the entry into MESGEN on the STICK (to check "Good News") interval, every 400 milliseconds. Section 4 provides periodic entry (every 100 milliseconds) into the RTABLE subroutine at each of the nodes. Sections 3 and 4 use portions of Section 2's code in order to reduce redundancy of code. Initialization of all the algorithm's tables, arrays, constants, and flags are accomplished by Section 5. When topological changes are required, Section 6 is added. The code of Sections 1 through 5 remains the same throughout all the simulation runs. It is Section 6 that gets changed whenever a topological change is desired. A more detailed discussion about the way the changes are accomplished is given in a later section of this chapter. Section 7, as in the baseline program, is the simulation timing control section. As explained in Chapter VI, it controls the traffic volume intensity levels and statistical output. The order of discussion for the sections is Section 5, Section 2, Section 3,

Section 4, and Section 1.

As stated earlier, Section 5 initializes all the tables, arrays, constants and flags used by the algorithm as a whole by activating the INITIALIZE, INITIALIZE\_D, COMPUTE\_D, and RTABLE subroutines for each of the nodes. The GENERATE instruction generates only one transaction (packet). The purpose of the packet is to move through the various HELP instructions activating the subroutine whose name is the A operand of the instruction. The B through G operands (P1 to P6 are the values of the parameters 1 to 6 of the packet but are used only for dummy arguments because of the instruction format. The moves through the HELP instructions starts at those given for Albany and terminates at the TERMINATE instruction after activating Tinker's initializing subroutines. The packet does not move on to the next HELP instruction until the present subroutine that it has activated is completed.

Section 2 gives the code necessary to enter MESGEN under the FTICK periodic interval. The code denoted by the encircled "A" generates a packet every 103 milliseconds, assigns priority status (parameter 2 = 2) and assigns the special status packet length of 389 bits (parameter 3 = 3) to the packet. The reason for generating a packet every 103 msec instead of exactly 100 msec is to help ensure that not more than one status packet is being processed by the same section of code at the same time. The packet enters the SPLIT block labeled ALBME where a duplicate packet



with the same parameter values is generated. One packet goes to the label ANDME (Andrews' equivalent MESGEN code). The first instruction at each of the other node's MESGEN code is a split instruction which sends another copy of the packet to the next succeeding node's MESGEN code for their processing. In this fashion, the MESGEN section of simulation code at each node is executed almost simultaneously. SAVEVALUE 122 is set to a value of one (denoting "BAD NEWS" processing in Albany's MESGEN subroutine ALMSGN). The sequence of instructions QUEUE, SEIZE, DEPART, and RELEASE is used to ensure the execution of the code denoted by the encircled "B" by only one status packet at a time. The packet has Albany assigned as the source node (parameter 4 = 1) and SAVEVALUE 125 is set to the value 0. The HELPC ALMSGN block is entered next to activate Albany's MESGEN subroutine. If the subroutine has in fact generated a status packet, (SAVEVALUES 57 - 64 contain Albany's status packet information), then the value of SAVEVALUE 125 is changed from its previous value of 0 to a value of 1. If ALMSGN has changed the value of SAVEVALUE 125, the TEST L instruction sends the packet to the instruction labeled ALBNY. If no status packet information was generated by ALMSGN, the packet terminates. Whenever the MESGEN determines a status packet is necessary, SAVEVALUES are used to store the actual information. The status information is not actually assigned to the packet which activated the MESGEN subroutine. The packets purpose is only to activate the appropriate subroutine

to manipulate (read or modify) the contents of the SAVE-VALUES which represent status packet information. Assume status packet information is generated, the packet enters the instruction labeled ALBNY where a duplicate copy is sent to the instruction labeled ANDW1. The original packet falls through to the ASSIGN statement where parameter 1 is loaded with Albany's ID as the destination node. At ALME1, the sequence of instructions QUEUE, SEIZE, DEPART, and RELEASE is used to ensure the execution of the code denoted by the encircled "C" and is executed by only one packet at a time. Because status information was generated, the algorithm's database must be updated to reflect the new current information. The HELPC ALUPDT instruction activates Albany's version of UPDATE to update Albany's LD matrix with the information contained in SAVEVALUES 57 - 64. The HELPA ALINID and HELPA ALCOMN recomputes Albany's D table and associated arrays. At ALME2, a similar instruction sequence to inhibit simultaneous processing of packets is executed. The packet enters the HELPC ALRTBL instruction which activates Albany's version of the RTABLE subroutine. Link routing arrays SROUTE and ROUTE are then recalculated. Having accomplished its intended purpose and no longer needed at the Albany node, the packet enters the TERMINATE block denoted by the encircled "D" and is removed from the system.

Albany has now processed its own status packet information but the other nodes also need to process Albany's status information. The SPLIT instruction labeled ALBME

had previously sent a packet to the instruction labeled ANDW1. In the code denoted by the encircled "E", a packet is generated for each of the other nodes with the appropriate destination node's ID number assigned to parameter 1 of that packet. Each packet is transferred to be input processed as any other packet is processed.

As indicated earlier in this section, MESGEN processing is identical for each of the nodes. The portion of code from ALBME down to and including the encircled "E" is duplicated for each node. The only changes in code are the SAVEVALUE instructions. Each node has its own set of SAVEVALUES. A complete listing of the SAVEVALUES and those Assigned to each node is given in Appendix E.

Section 3 enables the periodic entry into MESGEN for each of the nodes under STICK time interval. In order to reduce the chances of attempting to process two status packets at the same time, packets are generated every 401 msec instead of the advertised 400 msec. As in the FTICK code, the packet is assigned priority status (parameter 2 = 2) and the unique packet length of 389 bits (parameter 3 = 3). At ALBY (for Albany), a duplicate packet is sent to ANDE (for Andrews) while the original packet falls through into the SAVEVALUE instruction. SAVEVALUE 122 is set to zero to denote "GOOD NEWS" processing at Albany and the packet is transferred to ALB1 (in the previous FTICK coding). The processing of the packet is identical to that of the previous packet processing discussion only the SAVEVALUE 122

value of 0 causes the MESGEN code to check for new operational links and the absence of link congestion or node saturation (GOOD NEWS). The rest of the code in Section 3 sets the appropriate SAVEVALUES to zero for each of the respective nodes then transfers the packets to the proper entry point in each node's FTICK coding.

Periodic entry into the RTABLE subroutines at the various nodes is accomplished in Section 4. Packets are generated every 100 milliseconds and since these packets are not transmitted no values are assigned to any of their parameters. The sole purpose of a packet generated in this section is to cause the activation of a node's RTABLE subroutine. After generation, a packet enters the SPLIT instruction labeled ALBN (for Albany) whereupon a duplicate copy is sent to ANDS (for Andrews). The original packet falls through to the TRANSFER instruction and is transferred to ALME2 (in the previous FTICK code). The rest of Section 4's code creates and transfers packets to the respective entry points in that node's code to activate the appropriate RTABLE subroutine.

Now that the activation of the algorithm's Background subroutines have been discussed, changes to the main simulation program, Section 1, will be explained. Since each node requires the same general changes, the changes necessary for Albany's code are explained. The first required change comes as a requirement to activate Albany's PAKROUTE subroutine. The encircled "F" shows the code necessary to activate and use the results of the subroutine P1 through P6

(parameters 1 through 6 of the packet), which contain all the necessary information needed by the subroutine to select the trunk over which to route the packet. When the packet enters the HELPC ALPKRT instruction, the subroutine uses the values of P1 (destination node ID) and P4 (source node ID) in the calculation to select the best trunk. The selected trunk's ID number is stored in the FULLWORD SAVEVALUE 1. When control is returned to the GPSS simulation program, the packet enters the following ASSIGN block. The trunk ID number in FULLWORD SAVEVALUE 1 is assigned to packet parameter 5. The TRANSFER block uses the value of parameter 5 and FUNCTION ROUT1 to transfer the packet to the correct trunk queue.

Changes are also required to the ROUT1 function, encircled "G". Packets can be sent to any one of Albany's seven trunks, terminated locally or be removed from the system because the packet is undeliverable (the destination node of the packet is non-operational). Thus nine entries are required in the function. PAKROUTE returns one of nine numbers (1 - 9) to be placed in parameter 5 of the packet. If the packet value is 1 through 7, the packet is sent to the indicated trunk for transmission. If the value is an 8, the packet is sent to TERML for local termination processing, and if the value is a 9, the packet is undeliverable and is sent to a terminate block for removal from the system. Local termination processing, indicated by the encircled "H", includes changes to the program to check and see if the packet represents a status packet. The packet's length is checked

to see if the length corresponds to the length of a status packet. If it does, the packet is transferred to ALMB1 where the packet causes activation of Albany's UPDATE subroutine. Inside of Albany's UPDATE subroutine, ALUPDT, the correct set of SAVEVALUES, containing the new status information for the node whose status packet is being processed, is accomplished by a calculation using the packet's source node ID number. After UPDATE, the same packet causes execution of INITIALIZE\_D, COMPUTE\_D, and RTABLE to recalculate and update the algorithm's database then terminates.

Since status packets are being transmitted through the network, changes are required to the portion of code representing the transmission of packets. For example, a status packet from Albany is to be transmitted to Ft. Detrick (denoted by the encircled "I"). The packet enters the queue ALQ22, seizes the trunk ALTK3, and departs the queue. To determine how much transmission time is required for the packet, parameter 3 (containing the length of the packet) and FUNCTION ALF22 (denoted by the encircled "J") is used. Since the value of parameter 3 is a 3, the packet is sent to ALC22 where the packet accumulates 16 msec of transmission time. After acquiring the specified amount of time, the packet releases the trunk, ALTK3, and is transferred to the second SCM processor at Ft. Detrick (FTDE2) for further processing.

### Procedures for Running the Modified Program

The first set of four runs is with the network totally operational, that is, all trunks, links, and nodes up and functioning. The same method of running the baseline program with one run per traffic intensity spike of 60%, 70%, 80%, and 90% of network saturation is used for running the algorithm version of the simulation program. Appendix F shows the graphical and tabular results of this first set of simulation runs.

The program code used to simulate the network losses is the portion of the main program code with the comment title, "CODE FOR ENTRY INTO CONNECTIVITY CHANGE" (Section 6), and the algorithm's CONN\_CHANGE subroutine. A discussion is given detailing the program changes required for each of the different types of network losses. For ease of explanation, one specific example is discussed in each of the three types of topological changes.

Trunk Loss Simulation. Figure 26 shows the code of the main program necessary to drop the third trunk in the Albany - Ft. Detrick link. There are two parts of code in Figure 26. Part 1 generates one packet 25 seconds into the simulation. The packet is assigned priority status and the length of a status packet. An identical packet, including the same parameter values, is created in the SPLIT block (denoted by the encircled "A") and sent to label SECPR. The original packet falls

through into the SAVEVALUE instructions where SAVEVALUES 121, 126, and 127 are loaded with values used in ALCCHG, Albany's CONN\_CHANGE subroutine (Figure 27). Zero is loaded into SAVEVALUES 121 and 126 to set flags to denote a change in trunk status and that a trunk is to be dropped respectively. One in SAVEVALUE 122 sets the flag to denote "BAD NEWS" processing is to occur in Albany's version of MESGEN after the return from ALCCHG. A two in SAVEVALUE 127 indicates that a trunk will be dropped in Albany's link 2 to Ft. Detrick. The HELPC ALCCHG instruction causes control to be passed to the ALCCHG subroutine. In the subroutine, control is passed from statement 4 to statement 100. The ALLINS array is used to store the number of trunks in a particular link indexed by that link number. Since SAVEVALUE 126 has a 0 in it, control is passed to the next statement where a 2 is stored into ALLINS (2). Originally there were three trunks in link 2 to Ft. Detrick, but now there are two, in effect dropping a trunk from the link. Control is returned to the GPSS simulation program and the packet is transferred to a portion of code in Section 2. MESGEN is entered to see if dropping of the trunk requires the generation of new status information. If not, the packet is removed from the system. Otherwise its further processing is as described earlier in reference to the processing of a status packet. Similar processing goes on starting at label SECPR, encircled "B", Figure 26, to simulate Ft. Detrick's dropping of a trunk in its link to Albany.



# CODE FOR ENTRY INTO CONNECTIVITY CHANGE

FOR THIS SIMULATION, CONN-CHANGE IS USED TO CHANGE THE STATUS OF TRUNKS OR LINKS DEPENDING ON THE CODE BELOW.

FOR TRUNK LOSS, A SAVEVALUE FOR A PARTICULAR NODE IS SET WITH THE LINK # IN WHICH A PARTICULAR TRUNK IS TO BE CHANGED. ANOTHER SAVEVALUE IS SET AS A FLAG TO DROP THE TRUNK OR ADD THE TRUNK BACK IF THE FLAG IS 0 OR 1 RESPECTIVELY.

FOR LINK LOSS, ANOTHER SAVEVALUE IS SET TO THE NUMERICAL NUMBER OF THE LINK TO BE CHANGED. DEPENDING ON THE SAVEVALUE USED TO FLAG LINK CHANGE, THE LINK IS DROPPED OR ADDED BACK IF THAT FLAG IS 0 OR 1 RESPECTIVELY.

THE COMMENTS ON THE INSTRUCTIONS THEMSELVES BETTER CLARIFY WHAT ACTION IS TAKING PLACE.

IN AN EFFORT TO SAVE CODE, THE SAME SAVEVALUE THAT IS USED FOR LINK LOSS, IF SET TO 0, IS USED AS A FLAG TO SIGNIFY ENTRY INTO CONN-CHANGE TO PROCESS A TRUNK CHANGE.

A THIRD SAVEVALUE IS USED TO SIGNAL THE PARTICULAR TYPE OF PROCESSING IN MESSAGE TO BE ACCOMPLISHED.

NOTE: IN REALITY, A TRUNK IS FULL DUPLEX, THAT IS, TRAFFIC PASSES COMING AND GOING ON THE SAME TRUNK. IN THE SIMULATION PROGRAM, EACH SIMULATED TRUNK IS HALF DUPLEX. THEREFORE, RESPECTIVE CONN-CHANGE SURROUTINES MUST BE ENTERED AT BOTH NODES WHO ARE CONNECTED TOGETHER BY THE "TRUNK" OR LINK IN QUESTION.

1ST EXAMPLE - DROP 4TH TRUNK AT ALBANY AND THE CORRESPONDING TRUNK AT FT. DETRICK.

GENERATE 25000,,25002,1,,F GENERATE 1 PACKET 25 SECS INTO SIMULATION.  
 PRIORITY 2 GIVE IT PRIORITY CLASSIFICATION.  
 ASSIGN 2,2 SET PARAMETER 2 WITH THE PACKET PRIORITY.  
 ASSIGN 3,3 SET PARA. 3 WITH THE SPECIAL STAT. PKT LENGTH.  
 SPLIT 1,SECRP SEND COPY FOR 2ND NODE PROCESSING.  
 SAVEVALUE 121,0,XL SET FLAG FOR TRUNK LOSS.  
 SAVEVALUE 122,1,XL SET FLAG FOR 4TH NEWS PROCESSING.  
 SAVEVALUE 126,0,XL FLAG TO DROP TRUNK.  
 SAVEVALUE 127,2,XL DEOP TRUNK 4 AT ALBANY.  
 HELDPC ALCHG:P1,P2,P3,P4,P,P. GO DROP TRUNK.  
 TRANSFER ALBMY GO SET IS STAT. MSG. IS NECESSARY.  
 SECRP SAVEVALUE 140,0,XL SET FLAG FOR TRUNK LOSS.  
 SAVEVALUE 141,1,XL SET FLAG FOR 4TH NEWS PROCESSING.  
 SAVEVALUE 146,0,XL FLAG TO DROP TRUNK.  
 SAVEVALUE 147,1,XL DEOP TRUNK 5 AT FT. DETRICK.

## Section 6 Part 1

Fig 26. Code Necessary to Drop Trunk in Albany - Ft. Detrick Link

```

HELPG  DECHG,P1,P2,P3,P4,P5,P6  GO DRUP TRUNK.
TRANSFER ,DETM1  GO SEE IF STAT. MSG. IS NECESSARY.

*
*
*
*
Part 2
- REESTABLISH 3RD TRUNK AT ALBANY AND THE CORRESPONDING
  TRUNK AT FT. DETROIT.

GENERATE 46000,,46002,1,,,F GEN. 1 PACKET 46 SECONDS INTO SIMULATION.
PRIORITY 2  GIVE IT PRIORITY CLASSIFICATION.
ASSIGN 2+2  SET PARAMETER 2 WITH PRIORITY.
ASSIGN 3+3  SET PARAMETER 3 WITH SPECIAL STAT. MSG. LENGTH.
SPLIT 1,SECP1  SEND COPY FOR 2ND MODE PROCESSING.
SAVEVALUE 121,0,XL  SET FLAG FOR TRUNK ADD.
SAVEVALUE 122,0,XL  SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 126,1,XL  FLAG TO ADD TRUNK BACK.
HELPG  ALCHG,P1,P2,P3,P4,P5,P6  GO ADD TRUNK.
TRANSFER ,ALEM1  GO SET IF STAT. MSG. IS NECESSARY.
SECPI 1 140,0,XL  SET FLAG FOR TRUNK ADD.
SAVEVALUE 141,0,XL  SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 146,1,XL  FLAG TO ADD TRUNK BACK.
HELPG  DECHG,P1,P2,P3,P4,P5,P6  GO ADD TRUNK.
TRANSFER ,DETM1  GO SEE IF STAT. MSG. IS NECESSARY.

```

Fig 26. (CONT'D)



In order to bring the trunks back up to operational status at Albany and Ft. Detrick, the code in Part 2, Figure 26, is exercised. Forty-six seconds into the simulation, a packet is generated (the trunks have been down for twenty-one seconds). The packet is assigned priority status, given status packet length, and a duplicate copy created and sent to label SECPl. The only change in the SAVEVALUES is to SAVEVALUE 122 and 126. SAVEVALUE 122 is loaded with a zero to denote "GOOD NEWS" processing is to occur when MESGEN is entered in later code. SAVEVALUE 126 is loaded with a one to flag, in the ALCCHG subroutine, the reestablishment of the dropped trunk back into the link between Albany and Ft. Detrick. SAVEVALUES retain their values throughout and the simulation unless specifically changed, hence no need to change SAVEVALUE 127. When the ALCCHG subroutine is activated by the packet entering the HELPC ALCCHG instruction, the values of ALLINS (2) is changed back to a three, essentially bringing the trunk back up to operational status. When control returns to the GPSS simulation the packet transfers to the same portion of code in Section 2 and is processed similarly as described earlier.

The duplicate copy of the packet sent to SECPl, encircled "C", Figure 26, accomplishes for Ft. Detrick the same job as the packet did for Albany. The trunk at Ft. Detrick returns as one of the three operational trunks in the link to Albany and Ft. Detrick's version of MESGEN is entered to determine if new status information should be generated.

Link Loss Simulation. In this set of examples, program changes to Section 6 of the main program are made to facilitate simulating the loss of a link between two nodes. Again, for use of explanation, the specific example of losing the link connecting Gentile and Ft. Detrick is used. Figure 28 shows the code necessary to simulate the condition. As in the trunk example, there are two parts to Section 6. Part 1 is the code used in simulating the loss of the link, and Part 2 is the code used to bring the link back up. The packet is processed in similar fashion as described in the trunk loss example above with exceptions in values stored in SAVEVALUES. Gentile's version of CONN\_CHANGE (GECCHG), uses the values in SAVEVALUES 150, 151, 155, and 156. A one in SAVEVALUE 150 flags the section of code in GECCHG necessary to provide a change in link status. A one in SAVEVALUE 151 flags the future execution of MESGEN to check for "BAD NEWS". A zero in SAVEVALUE 155 flags the execution of code in GECCHG to drop the link whose value is in SAVEVALUE 156 (1 => Gentile's first link). When GECCHG (Figure 29), is activated, statement 4 passes control to statement 5. Indexes into Gentile's LD table (GEN1 and GEN2), are set up with GEN1 taking on Gentile's ID number and GEN2 taking on Ft. Detrick's ID number. At the encircled "A" of Figure 29, execution of the instruction determines whether the link change involves dropping the link from or adding it back to the Gentile node. If the link is to be dropped, (SAVEVALUE 155 = 0), control is passed to statement 10, other-

5TH EXAMPLE - DROP 1ST LINK AT GENTILE AND THE CORRESPONDING LINK  
AT FT. DETRICK.

Section 6  
Part 1

```

GENERATE 25000,,25002,1,,F GENERATE 1 PACKET 25 SECS INTO SIMULATION.
PRIORITY 2 GIVE IT PRIORITY CLASSIFICATION.
ASSIGN 2,2 SET PARAMETER 2 WITH THE PACKET PRIORITY.
ASSIGN 3,3 SET PARA. 3 WITH THE SPECIAL STAT. PKT LENGTH.
SPLIT 1,SECP1 SEND COPY FOR 2ND NODE PROCESSING.
SAVEVALUE 150,1,XL SET FLAG FOR LINK LOSS.
SAVEVALUE 151,1,XL SET FLAG FOR RAD NEWS PROCESSING.
SAVEVALUE 155,0,XL FLAG TO DROP LINK.
SAVEVALUE 156,1,XL DROP LINK 1 AT GENTILE.
HELPC GECHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER ,GENM1 GO SEE IF STAT. MSG. IS NECESSARY.
SECPR 140,1,XL SET FLAG FOR LINK LOSS.
SAVEVALUE 141,1,XL SET FLAG FOR RAD NEWS PROCESSING.
SAVEVALUE 140,0,XL FLAG TO DROP LINK.
SAVEVALUE 147,3,XL DROP LINK 3 AT FT. DETRICK.
HELPC DECCHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER ,DETM1 GO SEE IF STAT. MSG. IS NECESSARY.

```

Section 6  
Part 2

```

GENERATE 46000,,46002,1,,F GEN. 1 PACKET 46 SECONDS INTO SIMULATION.
PRIORITY 2 GIVE IT PRIORITY CLASSIFICATION.
ASSIGN 2,2 SET PARAMETER 2 WITH PRIORITY.
ASSIGN 3,3 SET PARA. 3 WITH SPECIAL STAT. MSG. LENGTH.
SPLIT 1,SECP1 SEND COPY FOR 2ND NODE PROCESSING.
SAVEVALUE 150,1,XL SET FLAG FOR LINK ADD.
SAVEVALUE 151,0,XL SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 155,1,XL FLAG TO ADD LINK RACK.
HELPC GECHG,P1,P2,P3,P4,P5,P6 GO ADD LINK.
TRANSFER ,GENM1 GO SEE IF STAT. MSG. IS NECESSARY.
SECPR1 140,1,XL SET FLAG FOR LINK ADD.
SAVEVALUE 141,0,XL SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 140,1,XL FLAG TO ADD LINK RACK.
HELPC DECCHG,P1,P2,P3,P4,P5,P6 GO ADD LINK.
TRANSFER ,DETM1 GO SEE IF STAT. MSG. IS NECESSARY.

```

Section 6  
Part 3

```

- REESTABLISH 1ST LINK AT GENTILE AND THE CORRESPONDING
LINK AT FT. DETRICK.

```

Fig 28. Code for Loss of Gentile - Ft. Detrick Link

```

SUBROUTINE RECCHG(PAR4VA,BLD,VAR,FUN,BVA,CHA,FAC,GKO,HSA,LOG,
+QUE,STO,TAT,FMA,FSA,HMA,HMA,LMA,MSA,SAVAGE)
C
C CONN-CHANGE UPDATES THE 40 TABLE TO REFLECT IF A LINK HAS
C COME UP OR SOME DOWN. THE APPROPRIATE LINK CONNECTIVITY
C ENTRY FOR INJ,N2) IS UPDATED TO DENOTE THE NEW CONNECTIVITY
C STATUS OF THAT LINK. MESSIN AND UPDATE (WHICH CALL MINHOP
C AND RTAPLE) SHOULD BE CALLED TO REFLECT THIS NEW LINK
C CONNECTIVITY STATUS.
C
INTEGER PAR4VA(4),BLD(2,4),VAR(1),FUN(2,4),BVA(1),CHA(3,1)
INTEGER FAC(4,1),GKO(1),HSA(1),LOG(1),QUE(4,1),STU(7,1),TAB(9,1)
INTEGER FMA(1),FSA(1),HMA(1),HMA(1),LMA(1),MSA(1)
REAL SAVAGE(1)
COMMON/GETA/LGELD(5,1),GEP(9),INF1,GENN,GENC(32),GEROUT(4),
+GL(1,3,1),GEHOP(8),GELIST(32),GESCLF,GENLN,GES,GES,GESNOT(9),
+GELINE(5),GERBUFF(1),GERPOT(9),GERFID(1),GEPARM(6),GERUP(5),
+GEZONE,SEI1,GEI2,GEIJS(1)
INTEGER GES,GENC,GEPI,GEN4,GEHOP,GEHOP,GEHOP,GEHOP,GEHOP,GEN1,
+GESELF,GEALF,GESENOT,GESENOT,GESENOT,GESENOT,GESENOT,GESENOT,
+GLA2,SEI3,GEIJS,GEIJS,GEIJS,GEIJS,GEIJS,GEIJS,GEIJS,GEIJS
REAL INPI
IF (INT(SAVAGE(156))) .NE. 0) GO TO 1
GO TO 10
GLA1 = GESELF
INDEX = GEP(GESELF) - 1
GEI2 = IABS(GENC(INDEX + (INT(SAVAGE(156)))))
IF (INT(SAVAGE(156))) .GT. 0) GO TO 15
GEI2AT = -1
GEI2AT = 1
GEI2UP(INT(SAVAGE(156))) = 1000
GO TO 21
GEI2AT = 1
GEI2UP(INT(SAVAGE(156))) = 1
GEI2UP(INT(SAVAGE(156))) = GEI2AT * ABS(GELD(GEN1,GEN2))
I = GEP(GEN1) - 1
I = I + 1
GEN3 = IABS(GENC(I))
IF (GEN2 .NE. GEN3) GO TO 25
GEI2(I) = GEI2AT + GEI3
GO TO 10
IF (INT(SAVAGE(156))) .GT. 0) GO TO 110
GO TO 110
GELINE(INTR(SAVAGE(156))) = 2
GO TO 110
RETURN
END

```

Fig 29. CONN\_CHANGE Subroutine for Gentile

wise control is passed to statement 15. At statement 10 GESTAT is set to a -1 (used shortly to set the appropriate entry in Gentile's LD matrix and the link number in Gentile's NC array to negative values). XXBUP (GEBUP for Gentile) is an array denoting the current status of the Gentile links. A one in the array indicates the link is up and a 1000 indicates the link is down. Since SAVEVALUE 156's value is a 1, GEBUP(1) is set to 1000 to indicate the loss of the link. (The GEBUP array is used in MESGEN to compare it against the previous status of the links contained in the XXSTUS array (GESTUS for Gentile)). Control is then passed to statement 20 where the value of the LD matrix indexed by GEN1 and GEN2 is made negative (used later in GEINID and GECOMD to recalculate the D matrix for Gentile). Index I value is calculated to equal the value of GEN2 and the corresponding value of Gentile's array is made negative. Control is then returned to the GPSS simulation program. In order to compensate for this new condition, the packet is transferred to a portion of code in Section 2 (GENM1) used to handle the processing of Gentile's version of MESGEN and Gentile's association entries into UPDATE, INITIAL\_D, COMPUTE\_D, and RTABLE. The duplicate copy of the packet sent to SECPR (encircled "A" on Figure 28), receives similar processing for the Ft. Detrick node as discussed above for the Gentile node.

After forty-six seconds of simulation elapses, the link between Gentile and Ft. Detrick is brought back up into operational status by the execution of the code in Part 2 of



Figure 28. SAVEVALUES are changed to flag the execution of code in GECCHG and DECCHG to reverse their previous actions and reestablish the link between Gentile and Ft. Detrick. CONN\_CHANGE is entered to bring the link back up, MESGEN builds new status information which is sent locally and to the other nodes. UPDATE makes the appropriate changes to the LD matrix, INITIALIZE\_D reinitializes the D matrix and associated arrays, COMPUTE\_D recomputes the new D matrix and RTABLE recomputes the link routing arrays SROUTE and ROUTE. The processing necessary to drop a link versus what is necessary to drop a trunk is more complicated because a major change in network connectivity is occurring. New status information will be generated when MESGEN is entered later or thus everything must be set up for other subroutines to simulate the condition correctly.

Node Loss Simulation. The code necessary to accomplish the removal of a node from the network is more complicated than that of either of the two previous examples. Each of the nodes connected to the non-operational node must remove their respective link to the down node from operational status and throw away the packets whose destination is the downed node. In this simulation program, even though the last node is disconnected from the rest of the network it still generates packets and processes them as if it were operational. It was easier to incorporate this change than remove the node's code completely when it goes down and replace the code when the node comes back up again. When the

node's PAKROUTE subroutine is activated, instead of putting the appropriate trunk number into the related FULLWORD SAVEVALUE an integer number is placed in the related FULLWORD SAVEVALUE which, when control is returned to the main simulation program, causes the packet to be sent to a TERMINATE block for removal from the system.

Similar changes were required in each of the other node's PAKROUTE subroutines. If the packet's destination is the downed node, the appropriate integer number is placed into the related FULLWORD SAVEVALUE which will result in the packets eventual removal from the system. Figure 30 shows Section 6's code required to simulate the loss and reestablishment of the Ft. Detrick node. As in the previous two examples there are two parts to the code. The nodes connected to Ft. Detrick (Albany, Andrews, Hancock, Gentile, and Tinker) each have their own section of code in Part 1 to simulate the loss of Ft. Detrick. In each node's code, their respective SAVEVALUES are set up to drop their link to Ft. Detrick and CONN\_CHANGE is executed at each node. Ft. Detrick's code (starting at the label SECPD) loads its SAVEVALUE with special flags. SAVEVALUE 141 is loaded with the normal value of 1 to denote "BAD NEWS" processing in MESGEN. The other SAVEVALUES are loaded with dummy values which, in effect, causes no change in the status of any of its links in DECCHG. The key SAVEVALUE is 146. It is loaded with a value of 1. Its purpose will be explained shortly.

When CONN\_CHANGE has been executed at each connected

Section 1  
Part 1

16TH EXAMPLE - DROP NODE FT. DETRICK FROM THE NETWORK.

GENERATE	25000,25002,1,,F GEN. 1 PACKET 25 SECONDS INTO SIMULATION.
PRIORITY	2
ASSIGN	2,2
ASSIGN	3,3
SPLIT	1,SECPD
SAVEVALUE	121,1,XL
SAVEVALUE	122,1,XL
SAVEVALUE	126,0,XL
SAVEVALUE	127,2,XL
HELPC	ALCHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER	,ALCHG1
SPLIT	1,SECPA
SAVEVALUE	140,1,XL
SAVEVALUE	141,1,XL
SAVEVALUE	140,1,XL
SAVEVALUE	147,1,XL
HELPC	DECHG,P1,P2,P3,P4,P5,P6 GO DROP THE NODE.
TRANSFER	,DETH1
SPLIT	1,SECPH
SAVEVALUE	130,1,XL
SAVEVALUE	131,1,XL
SAVEVALUE	135,0,XL
SAVEVALUE	136,2,XL
HELPC	ANCHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER	,ANDH1
SPLIT	1,SECPG
SAVEVALUE	160,1,XL
SAVEVALUE	161,1,XL
SAVEVALUE	164,0,XL
SAVEVALUE	165,2,XL
HELPC	HACHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER	,HAMH1
SPLIT	1,SECT
SAVEVALUE	150,1,XL
SAVEVALUE	151,1,XL
SAVEVALUE	155,0,XL
SAVEVALUE	156,1,XL
HELPC	GECHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER	,GEMH1
SPLIT	1,SECT
SAVEVALUE	190,1,XL
SAVEVALUE	191,1,XL
SAVEVALUE	190,6,XL

GO SEE IF STAT. MSG. IS NECESSARY.  
SEND COPY FOR ANDREWS PROCESSING.  
SET FLAG FOR LINK CHANGE.  
SET FLAG FOR BAD NEWS PROCESSING.  
SET FLAG FOR FT. DETRICK'S LOSS.  
SET DUMMY FLAG FOR ALBANY'S LINK.  
GO SEE IF STAT. MSG. IS NECESSARY.  
SEND COPY FOR HANCOCK PROCESSING.  
SET FLAG FOR LINK CHANGE.  
SET FLAG FOR BAD NEWS PROCESSING.  
FLAG TO DROP LINK.  
DROP LINK 2 AT ANDREWS.  
GO SEE IF STAT. MSG. IS NECESSARY.  
SEND COPY FOR GENTILE PROCESSING.  
SET FLAG FOR LINK CHANGE.  
SET FLAG FOR BAD NEWS PROCESSING.  
FLAG TO DROP LINK.  
DROP LINK 2 AT HANCOCK.  
GO SEE IF STAT. MSG. IS NECESSARY.  
SEND COPY FOR LINKER PROCESSING.  
SET FLAG FOR LINK CHANGE.  
SET FLAG FOR BAD NEWS PROCESSING.  
FLAG TO DROP LINK.  
DROP LINK 1 AT GENTILE.  
GO SEE IF STAT. MSG. IS NECESSARY.  
SET FLAG FOR LINK CHANGE.  
SET FLAG FOR BAD NEWS PROCESSING.  
FLAG TO DROP LINK.

Fig 30. Example of Code for Loss and Return of a Node

```

SAVEVALUE 197,3,XL      DROP LINK 3 AT TINKER.
HELPC      TICCHG,P1,P2,P3,P4,P5,P6 GO DROP LINK.
TRANSFER   ,TINH1      GO SEE IF STAT. MSG. IS NECESSARY.

```

RESTORE THE FT. DETRICK NODE TO OPERATIONAL CAPABILITY.

```

Part 2
GENERATE 46300,,46002,1,,F GEN. 1 PACKET 46 SECONDS INTO SIMULATION.
PRIORITY 2      GIVE IT PRIORITY CLASSIFICATION.
ASSIGN    2,2    SET PARAMETER 2 WITH PRIORITY.
ASSIGN    3,3    SET PAPA. 3 WITH SPECIAL STAT. MSG. LENGTH.
SPLIT     1,SECP1
SAVEVALUE 121,1,XL      SEND COPY FOR FT.DETRICK PROCESSING.
SAVEVALUE 122,0,XL      SET FLAG FOR LINK CHANGE.
SAVEVALUE 126,1,XL      SET FLAG FOR GOOD NEWS PROCESSING.
HELPC     ALOCHG,P1,P2,P3,P4,P5,P6 GO ADD LINK.
TRANSFER   ,ALPH1      GO SEE IF STAT. MSG. IS NECESSARY.
SPLIT     1,SECP2      SEND COPY FOR ANDREWS' PROCESSING.
SAVEVALUE 140,1,XL      SET FLAG FOR LINK CHANGE.
SAVEVALUE 141,0,XL      SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 145,0,XL      RESET FLAG FOR DETRICK'S RETURN.
TRANSFER   ,DETH1      GO SEE IF STAT. MSG. IS NECESSARY.
SPLIT     1,SECP3      SEND COPY FOR HANCOCK PROCESSING.
SAVEVALUE 130,1,XL      SET FLAG FOR LINK CHANGE.
SAVEVALUE 131,0,XL      SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 135,1,XL      FLAG TO ADD LINK BACK.
HELPC     ANOCHG,P1,P2,P3,P4,P5,P6 GO ADD LINK.
TRANSFER   ,ANDH1      GO SEE IF STAT. MSG. IS NECESSARY.
SPLIT     1,SECP4      SEND COPY FOR GENTILE PROCESSING.
SAVEVALUE 100,1,XL      SET FLAG FOR LINK CHANGE.
SAVEVALUE 161,0,XL      SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 167,1,XL      FLAG TO ADD LINK BACK.
HELPC     HAUCHG,P1,P2,P3,P4,P5,P6 GO ADD LINK.
TRANSFER   ,HARM1      GO SEE IF STAT. MSG. IS NECESSARY.
SPLIT     1,SECP5      SEND COPY FOR TINKER PROCESSING.
SAVEVALUE 150,1,XL      SET FLAG FOR LINK CHANGE.
SAVEVALUE 151,0,XL      SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE 157,1,XL      FLAG TO ADD LINK BACK.
HELPC     GEOCHG,P1,P2,P3,P4,P5,P6 GO ADD LINK.
TRANSFER   ,GEMH1      GO SEE IF STAT. MSG. IS NECESSARY.
SECP5 SAVEVALUE 190,1,XL      SET FLAG FOR LINK CHANGE.

```

Fig 30. (CONT'D)

SAVEVALUE	191,0,XL	SET FLAG FOR GOOD NEWS PROCESSING.
SAVEVALUE	196,1,XL	FLAG TO ADD LINK BACK.
HELPC	TICCHG,P1,P2,P3,P4,P5,P6	GO ADD LINK.
TRANSFER	,TIM1	GO SEE IF STAT. MSG. IS NECESSARY.

Fig 30. (CONT'D)

node and at Ft. Detrick, the individual status are transferred to portions of code in Section 2 to activate the other subroutines necessary to complete the generation of new status information, update and recalculate the node's database and to broadcast each node's new status information to the rest of the network. After 46 seconds of simulation time, the GENERATE block in Part 2, generates 1 packet. The packet and its copies (created from the SPLIT blocks) process through their respective node's resetting the SAVEVALUES which are needed to simulate Ft. Detrick coming back up. CONN\_CHANGE is executed at each node except Ft. Detrick. Due to the dummy values in Ft. Detrick's SAVEVALUES, if DECCHG were executed, Ft. Detrick's link to Albany would be simulated as being down.

After return from CONN\_CHANGE, the various status packets are sent to portions of code in Section 2 for further processing necessary to recalculate and update their respective node's database.

The only program changes to the algorithm were to each node's version of PAKROUTE. The same four changes were required at every node's PAKROUTE subroutine except Ft. Detrick's which required three changes. Figure 31 shows the resulting PAKROUTE subroutine for Albany. The encircled "A" shows the four instruction changes. The first instruction tests to see if Ft. Detrick is down (SAVEVALUE 146 = 1). If the node is not down, control is passed to statement 3 and normal processing occurs. If Ft. Detrick is down, the next instruction tests to see if the destination of this packet is Ft. Detrick.



```

5      GO TO 1000
      IF (INDRC .EQ. ALSELF) GO TO 10
      K = ALPOUT(IDSTN)
      GO TO 15
10     K = ALSROT(IDSTN)
15     IF (K .NE. 0) GO TO 15
      FSA(1) = 9
      GO TO 1000
16     I = ALLINS(V)
      J = ALPIC(K)
      IF (I .NE. 1) GO TO 20
      FSA(1) = J
      GO TO 1000
20     IF (I .EQ. 2) GO TO 30
      IF (ALBUFT(J) .LE. ALBUFT(J+1)) GO TO 25
      FSA(1) = J + 1
      GO TO 1000
25     FSA(1) = J
      GO TO 1000
30     IF (ALBUFT(J) .GT. ALBUFT(J+1)) GO TO 40
      IF (ALBUFT(J) .GT. ALBUFT(J+2)) GO TO 35
      FSA(1) = J
      GO TO 1000
35     FSA(1) = J + 2
      GO TO 1000
40     IF (ALBUFT(J+1) .GT. ALBUFT(J+2)) GO TO 45
      FSA(1) = J + 1
      GO TO 1000
45     FSA(1) = J + 2
1000  RETURN
      END

```

Fig 31. (CONT'D)



(packet parameter 1 = 3). If the destination is not Ft. Detrick, control is passed to statement 3 for normal processing. If Ft. Detrick is the destination, FULLWORD SAVEVALUE 1 is set to 9. Control is then passed to statement 1000 for return to the GPSS program. Back in the simulation program the value in FULLWORD SAVEVALUE 1 is moved into packet parameter 5 and the packet is transferred to UNDEL for removal from the network.

Figure 32 shows the resultant code for Ft. Detrick. The code shown by the encircled "A" are the three changes added to the PAKROUTE subroutine at Ft. Detrick. If the node is not down, control is passed to statement 3 for normal processing. If the node is down, it can't transmit any packets, thus 13 is stored in FULLWORD SAVEVALUE 3. Control is passed to statement 1000 for return to the main simulation program. Upon return, the value in FULLWORD SAVEVALUE 3 is moved in packet parameter 5. The value of FULLWORD SAVEVALUE 3 (14) is used by FUNCTION ROUT3 to send the packet to UNDE3 for termination and removal from the system.

The following chapter contains an analysis of the algorithm's ability to adapt to traffic fluctuations while network topology changes are occurring. The chapter also contains a comparison summary of the packet delay results obtained by the simulation exercise and the packet delay requirements as set forth in DCA's Performance Specifications for the AUTODIN II network. The summary includes specific examples of topological changes and traffic fluctuation

```

SUBROUTINE DEPKRT(PAR1VA,BLO,VAR,FUN,BVA,CHA,FAC,GRO,MSA,LOG,
*QUE,STO,TAB,FMA,FSA,MMA,3MA,LMA,BSA,SAVVDE)
C
C PAKROUTE LOOKS AT THE QUEUE LENGTHS OF THE INDIVIDUAL TRUNKS
C MAKING UP THE LINK AS SELECTED BY SKROUTE OR ROUTE(SOURCE
C NODE OR TANDEM NODE) AND SELECTS THE TRUNK WITH THE LEAST
C NUMBER OF PACKETS QUEUED TO BE THE TRUNK TO QUEUE THE
C CURRENT PACKET ON. PASSED TO PAKROUTE ARE THE PARAMETERS
C (DEPARM) THAT BELONG TO THE CURRENT PACKET. THE PARAMETERS
C ARE RESPECTIVELY:
C
C PARAMETER 1: DESTINATION NODE ID# OF THE PACKET.
C " 2: PACKET PRIORITY 1-NON Prio, 2-PRIORITY
C " 3: PACKET LENGTH 1-SHORT, 2-LONG
C " 4: SOURCE NODE ID# OF THE PACKET.
C
C SAVVDE(3) WILL CONTAIN THE TRUNK ID # TO TRANSMIT
C THE PACKET ON.
C
C TRUNK ID = 13, LOCAL DELIVERY.
C TRUNK ID = 14, TERMINATE, UNDELIVERABLE.
C
C
C INTEGER PAR1VA(5),BLO(2,1),VAR(1),FUN(2,1),BVA(1),CHA(3,1)
C INTEGER FAC(5,1),GRO(1),MSA(1),LOG(1),QUE(4,1),STO(7,1),TAB(9,1)
C INTEGER FMA(1),FSA(1),MMA(1),BMA(1),LMA(1),BSA(1)
C REAL SAVVDE(1)
C
C COMMON/DETA/LDELD(3,3),DEP(9),INFI,DENN,DENCI(32),DEROUT(8),
C *DEJ(3,8),DEHOP(8),ULIST(32),DESELF,DEMLN,DESD,DESR(8),
C *DELI(5),DEBUFF(5),DETRFI(12),DEBFID(5),DEPARM(6),DEBUP(5),
C *DEZONE,DETI,DETR,DESTUS(3)
C
C INTEGER DES,DEN,DEP,DEMI,DEHOP,DELIST,DEBFID,DEBUP,DEZONE,
C *DESELF,DEMLN,DEE,DEROUT,DESDOT,DELI(5),DELI(5),DEBUFF,DEPARM,
C *DETI,DETR,DESTUS
C
C REAL INFI
C
C DO 1 I=1,6
C   DEPARM(1:DE) = PARHVA(INF)
C   CONTINUE
C
C DO 2 I=1,12
C   DEBUFT(I) = INT(SAVVDE(13+I))
C   CONTINUE
C
C IDSRC = DEP/RM(4)
C IUSTN = DEPARM(1)
C IF (INT(SAVVDE(146)) .NE. 1) GO TO 3
C FSA(3) = 14
C GO TO 1111
C IF (IDSRC .NE. DESELF) GO TO 5
C FSA(3) = 13
C GO TO 1111

```

Fig 32. PAKROUTE for Ft. Detrick

```

5  IF (IDSRC .EQ. DESELF) GO TO 10
    K = DEROUT(IDSTN)
    GO TO 15
10  K = DESROT(IDSTN)
15  IF (K .NE. 1) GO TO 15
    FSA(3) = 14
    GO TO 1010
16  I = DELINS(K)
    J = DEPRIC(K)
    IF (I .NE. 1) GO TO 20
    FSA(3) = J
    GO TO 1010
20  IF (I .EQ. 1) GO TO 30
    IF (DEBUFT(J) .LE. DEBUFT(J+1)) GO TO 25
    FSA(3) = J + 1
    GO TO 1010
25  FSA(3) = J
    GO TO 1010
30  IF (DEBUFT(J) .GT. DEBUFT(J+1)) GO TO 40
    IF (DEBUFT(J) .GT. DEBUFT(J+2)) GO TO 35
    FSA(3) = J
    GO TO 1010
35  FSA(3) = J + 2
    GO TO 1010
40  IF (DEBUFT(J+1) .GT. DEBUFT(J+2)) GO TO 45
    FSA(3) = J + 1
    GO TO 1010
45  FSA(3) = J + 2
1010 RETURN
      END

```

Fig 32. (CONT'D)

levels not satisfying the Performance Specifications based on statistical Hypothesis tests and confidence intervals, and a short section on the disturbance periods (the time between the traffic spike and the return of the network to the system mean delay). The paper concludes with recommendations on possible network reconfiguration changes and ideas on further simulation exercises.

## VIII Simulation Results and Analysis

It should first be stated that the development of the simulation model, and thus its packet delay results evolved from the author's interpretation and implementation of DCA's preliminary design requirements package which included traffic volume figures, internodal packet exchanges, and the network Base Routing matrix. Since the actual algorithm code was not available, the author's implementation of the algorithm in Fortran is also susceptible to interpretation. It is the author's opinion that given the model used and the methodology of its development, the conclusions (in Chapter IX), statements, and analysis of the delay results in this report are factual.

### Algorithm Adaptability

Webster's New Collegiate Dictionary defines adaptation as "the act or process taken to adjust to environmental conditions". The ability to adapt to these changing environmental conditions (traffic volume fluctuations and network topology) is a measure of the hierarchical routing algorithm's performance. If the algorithm causes undue excessive packet delay in the process of adapting to and compensating for changing environmental conditions, it is said to be not well-behaved. Without any precedence to go by in defining "undue, excessive packet delay", the author appends the following values to the definition; for any mean packet delay from the simulation equal to or greater

than 570 msec for short packets and 795 msec for long packets, the algorithm is said to be not well-behaved. Another measure would be 2025 msec for the maximum packet delay of short packets and 2250 msec for the maximum packet delay of long packets. These values (defined by the author) are one and one half times the packet delay requirements given in Figure 25, Chapter VI.

The author developed an additional performance criterion as a measure to determine another behavioral pattern of the algorithm. The "disturbance period" is defined to be the time it takes for the network to return to within a 95% Confidence Interval of the system mean packet delay after the occurrence of a traffic spike. The system mean packet delay is the delay value in parenthesis at the 30 second time frame on each graph. Also for identification purposes, the delay values in parentheses during disturbance periods, in graphs of Appendices F through I, are the maximum packet delays incurred from packets terminating (locally delivered) within the previous one second time interval.

Having defined two performance characteristics to determine the adaptability of the routing algorithm, four sample cases are discussed with respect to the algorithm's ability to adapt to the changing environmental conditions.

#### Selected Results Explained

The four cases being discussed are (1) the fully operational network with the algorithm versus the baseline

program, (2) an example from the trunk loss program runs, (3) an example from the link loss program runs, and (4) an example from the node loss program runs.

Fully Operational Network Versus Baseline. In this section, the graphs of the baseline results (Figures 22, 23 and 24 of Chapter VI) are compared against the graphs of the fully operational network in Appendix F (Figures F-1, F-2, and F-3).

In comparing the graphs of the system delay (all packets), Figures 22 and F-1, the baseline program takes nine seconds to return to equilibrium whereas the algorithm version takes only five seconds. These disturbance periods mean the algorithm can react to traffic fluctuations in intensity faster than the static routing methodology of the baseline program. Also shown in these graphs is the maximum delay times experienced by any packet during the disturbance period. The maximum delay a packet experiences in the algorithm version (1850 msec) occurs two seconds after the 90% spike. In the baseline program, the maximum delay (2830 msec) occurs three seconds after the 90% spike. The mean delays are higher in the algorithm version than in the baseline but this is due to additional load on the system because of the system status packets being generated and broadcast throughout the network. With the addition of the algorithm, the additional status packets add 6% to the traffic volume in the network over and above what the baseline was processing. Also as a matter of interest, is the fact that all the packets

in this additional 6% are priority packets which in effect delay the vast majority of the other packets in the network which are non-priority. But even with this additional traffic load, the algorithm minimized the affect of the traffic spike, lowered maximum delay times, and reacted to the traffic fluctuation better than the baseline program.

Figures 23 and F-2 are related graphs showing the system delay for non-priority - short packets. The same conclusions on the algorithm performance can be drawn if the two graphs are compared to each other. The disturbance period is shorter with the algorithm (5 seconds versus 7 seconds), maximum delay is lower (1850 msec versus 2710 msec), and the mean delays during the disturbance period is lower for the baseline. An interesting characteristic of the algorithm is the apparent immediate reaction to the traffic spike. One second after the spike, the maximum mean delay is reached, but in the baseline the maximum mean delay occurs two seconds after the spike.

Figures 24 and F-3 show the results of the two simulation versions for the system delay of non-priority - long packets. The affect of the algorithm on the disturbance period is pronounced in comparing the delay trends at the 90% saturation level (4 seconds in the algorithm version and 9 seconds in the baseline).

The data obtained for the short and long priority packets cannot really be compared. As stated in Chapter VI, there are not enough user priority packets generated for qualitative or quantitative analysis. The data does show



though, (in Tables IX and X of Chapter VI, and Tables F-1 and F-II) that the delays experienced by the priority packets did not exceed the mean or maximum delay constraints as specified in the DCA Performance Specifications (Figure 25, Chapter VI).

Simulated Topology Degradations. One of the characteristics of an efficient routing algorithm is its ability to recognize and respond in a prompt, orderly manner to network topology changes. These changes could consist of losses of trunks, complete links, or even nodes in the network. In this section of the chapter, the simulation of topological changes are made selectively, choosing one of the network's trunks, links, or nodes, and dropping it from the network. While the facility is non-operational, a traffic spike occurs, requiring the algorithm not only to compensate for the lost facility, but also to contend with the spiked increase in traffic volume. The selection of which trunks, links, and nodes to drop is made based on dropping a trunk in the most heavily utilized link, dropping that link itself, and dropping the most vital tandem nodes of the network. In order to have something to measure in the disturbance periods of the topological change simulation runs, the disturbance periods of the fully connected network simulation runs (Figures F-1 through F-3) are used as control samples. For the graphs showing packet delays for all packets (Figure F-1), the disturbance period is 5 seconds (from time frame 31 to time frame 36). For the graphs showing packet delays for short

packets (Figure F-2), the disturbance period is also five seconds. For the graphs showing packet delays for long packets (Figure F-3), the disturbance period is four seconds (from time frame 31 to time frame 35).

Sample Trunk Loss. The trunks dropped are the trunks in the links connecting Albany - Ft. Detrick, Ft. Detrick - Gentile, Andrews - Tinker, Ft. Detrick - Tinker, and Gentile - Tinker. Each of the five examples involves running the program four times, once per saturation spike. In the AUTODIN II network, trunks are full duplex (two-way) transmission lines, that is, traffic is both sent and received on the same trunk. However, in the simulation, two one-way trunks are required to effect the simulation of one full duplex trunk in the proposed network.

Results from simulation runs involving the losses of trunks are given in Appendix G. Each example of the graphical and tabular results is preceded by a page informing the reader which set of trunks is dropped for that example.

For the trunk loss example, the simulation runs simulating the loss of a trunk in the link between Gentile and Tinker (Figures G-13 through G-15), is selected. Although it is one of the worst-case examples for the set of trunk loss runs, the use of it in the ensuing discussion merits investigation. Even with the loss of the trunk, the disturbance periods for both the all-packet graph (Figure G-13), and the short packet graph (Figure G-14), equals the matching graphs of the control sample. The disturbance period for

the long packet graph (Figure G-15) is 5 seconds which is only one second longer than that of the long packet control sample. Typically, this means even with the additional burden of a lost primary trunk, the algorithm enables the network to withstand a large traffic spike (90% saturation worst-case) and still return to within a 95% Confidence Interval of the mean system delay within five seconds of the time of the spike. If another traffic spike occurs within five seconds of the first spike, the network would undergo additional stress and could quite possibly become not well-behaved. Since the network had already attained mean delay equilibrium by the time the trunk came back up, its addition to the network did very little to lower the system mean packet delay. An interesting aspect of the delay results during the disturbance period shows that the mean and maximum packet delays for the 60, 70, and 80 percent saturation runs (Figures G-13 through G-15) are less than those of the control sample (Figures F-1 through F-3). This could imply that the algorithm calculates better paths for packets to take while under moderate stress. Although the algorithm is well-behaved in this trunk loss sample, the mean and maximum packet delays for 90% exceeds the DCA delay requirements.

Sample Link Loss. The links simulated going down are the links connecting Albany - Ft. Detrick, Albany - Andrews, Gentile - Ft. Detrick, Andrews - Hancock, Andrews - Ft. Detrick, Andrews - Tinker, Ft. Detrick - Tinker, and Gentile - Tinker. Appendix H contains the graphical and tab-

ular results of the eight examples of link losses. They are arranged in a similar fashion as those given in Appendix G.

For an example of the link loss sample runs (Figures H-13 through H-15), the Andrews - Tinker link is chosen for discussion. The impairment to the network as a result of the lost link is evident in the graphs. In both the all-packet delay and the short packet delay graphs (Figures H-13 and H-14 respectively), the disturbance period for the 60, 70, and 80 percent saturation runs is seven seconds (from time frame 31 to time frame 38), but for the 90% run it is nine seconds. In the long packet delay graph (Figure H-15), the disturbance period is six seconds for the 60, 70, and 80 percent runs and nine seconds for the 90 percent run. The longer recovery times indicate a threat to the network's stability as the result of the loss of a primary link. Although well-behaved in the 60, 70, and 80 percent ranges, both the mean and maximum packet delays exceed those of a well-behaved algorithm for the 90 percent saturation level. When the link comes back up (time frame 42), its result on the network causes a slight smoothing effect on the packet delays.

Sample Node Loss. The nodes selected for removal from the network are Ft. Detrick, Gentile, and Tinker. Appendix I, in the same format as Appendices G and H, contain the graphical and tabular delay results for the node loss examples.

The removal of Ft. Detrick from the network is the exam-

ple from the set of node losses (Figures I-1 through I-3, Appendix I), to be discussed. The example shows the disastrous effects the removal of a primary node has on the network. On all three graphs the disturbance period is twelve seconds (one second after the node comes back up at time frame 42). The occurrence of any moderate traffic spike within the twelve second time period would most likely result in massive packet delays and a strong possibility of unstable and unbounded output trunk queues at the other network nodes. The algorithm is not well-behaved in any of the packet delay graphs, in fact, packets incur delays of up to four and a half seconds in the 90 percent run.

Due to the limited number of priority packets generated (1% of all packets), the tabular results of the priority packet delays in each set of examples discussed above does not provide conclusive evidence as to the performance of the algorithm for priority packets. Inspection of the results does show three things however; all of the mean and maximum delay values for short packets are within DCA requirements, about one and one half percent of the long packet mean delays exceed the DCA requirement of 330 msec, but none exceed the maximum delay requirement of 730 msec, and the topological changes seems to have no effect on the mean and maximum packet delays for priority packets.

#### Summary Tables

In order to consolidate and present all of the infor-

mation in Appendices F through I, in a more visible manner, this last section presents four tables to the reader. The thrusts of the four tables are to show the adaptability of the hierarchical routing algorithm through the two performance criteria defined previously and to back up the statistical results of the simulation runs. The statistical calculations provide 95% Confidence Intervals and One-Tail Null Hypothesis tests on the data contained in Appendices F through I on which the four tables are based.

Using a 95% Confidence Interval to determine when the network returned to equilibrium after the traffic intensity spike, Table XI gives a summary of the various disturbance periods (in seconds) for each simulation run conducted. The "All - Packets" graphs of each set of runs were used to generate Table XI.

Tables XII and XIII are produced using the Null Hypothesis Test and testing for  $\mu > \mu_0$  at  $\alpha = .05$  for non-priority short packets and non-priority long packets respectively. The Null Hypothesis was  $\mu = \mu_0$  ( $\mu_0 = 380$  msec for short packets,  $\mu_0 = 530$  msec for long packets), and the Alternate Hypothesis was  $\mu > \mu_0$ . The tests were conducted either assuming large sample populations with a normal distribution or known standard deviations from the output of each computer simulation run. The "dashed" entries correspond to where the Null Hypothesis was accepted and the "OVER" entries correspond to the rejection of the Null Hypothesis in favor of the Alternate Hypothesis.

TABLE XI

## Disturbance Period Summary

(entries are in seconds)

<u>Network Configuration</u>	Traffic Spike Level			
	<u>60%</u>	<u>70%</u>	<u>80%</u>	<u>90%</u>
<u>Fully Operational</u>	3	3	5	5
<u>Trunk Loss</u>				
Albany - Ft. Detrick	5	4	5	5
Andrews - Tinker	5	4	8	5
Ft. Detrick - Tinker	3	3	5	4
Ft. Detrick - Gentile	5	5	5	5
Gentile - Tinker	5	5	3	4
<u>Link Loss</u>				
Albany - Ft. Detrick	4	6	8	12
Albany - Andrews	4	4	5	4
Andrews - Hancock	3	4	5	4
Andrews - Ft. Detrick	3	5	10	10
Andrews - Tinker	3	7	7	10
Ft. Detrick - Gentile	7	5	7	7
Ft. Detrick - Tinker	3	4	3	4
Gentile - Tinker	6	3	6	6

TABLE XI (CONT'D)

(entries are in seconds)

Network Configuration

Traffic Spike Level			
<u>60%</u>	<u>70%</u>	<u>80%</u>	<u>90%</u>

Node Loss

Ft. Detrick

12      12      10      10

Gentile

5      4      4      6

Tinker

4      4      7      5



TABLE XII

## Excessive Mean Delay Summary for Non-Priority - Short Packets

<u>Network Configuration</u>	<u>Traffic Spike Level</u>		
	<u>60%</u>	<u>70%</u>	<u>80%</u>
<u>Fully Operational</u>	-	-	over
<u>Trunk Loss</u>			
Albany - Ft. Detrick	-	-	over
Andrews - Tinker	-	-	over
Ft. Detrick - Tinker	-	-	over
Ft. Detrick - Gentile	-	-	over
Gentile - Tinker	-	-	over
<u>Link Loss</u>			
Albany - Ft. Detrick	-	-	over
Albany - Andrews	-	-	over
Andrews - Hancock	-	-	over
Andrews - Ft. Detrick	-	-	over
Andrews - Tinker	-	over	over
Ft. Detrick - Gentile	-	-	over
Ft. Detrick - Tinker	-	-	over
Gentile - Tinker	-	-	over

TABLE XII (CONT'D)

<u>Network Configuration</u>	<u>Traffic Spike Level</u>		
	<u>60%</u>	<u>70%</u>	<u>80%</u>
<u>Node Loss</u>			
Ft. Detrick	over	over	over
Gentile	-	over	over
Tinker	-	-	over

TABLE XIII

Excessive Mean Delay Summary for Non-Priority - Long Packets

<u>Network Configuration</u>	<u>Traffic Spike Level</u>		
	<u>60%</u>	<u>70%</u>	<u>80%</u>
<u>Fully Operational</u>	-	-	-
<u>Trunk Loss</u>			
Albany - Ft. Detrick	-	-	over
Andrews - Tinker	-	-	over
Ft. Detrick - Tinker	-	-	over
Ft. Detrick - Gentile	-	-	over
Gentile - Tinker	-	-	over
<u>Link Loss</u>			
Albany - Ft. Detrick	-	-	over
Albany - Andrews	-	-	over
Andrews - Hancock	-	-	-
Andrews - Ft. Detrick	-	-	over
Andrews - Tinker	-	-	over
Ft. Detrick - Gentile	-	-	-
Ft. Detrick - Tinker	-	-	over
Gentile - Tinker	-	-	-

TABLE XIII (CONT'D)

<u>Network Configuration</u>	<u>Traffic Spike Level</u>			
	<u>60%</u>	<u>70%</u>	<u>80%</u>	<u>90%</u>
<u>Node Loss</u>				
Ft. Detrick	over	over	over	over
Gentile	-	-	-	over
Tinker	-	-	-	over

The last table, Table XIV, gives a summary of when the algorithm was not well-behaved as defined by the author previously. To restate, if a packet has a mean delay greater than or equal to 570 msec for short packets and 795 msec for long packets or has a maximum delay value greater than or equal to 2025 msec for short packets and 2250 msec for long packets, then the algorithm is said to be not well-behaved at the traffic level in which the excess delay occurred. The "dashed" entries represent when the algorithm was well-behaved and the "X" entries correspond to when the algorithm was not well-behaved.

The last chapter presents the conclusions and recommendations of this paper.

TABLE XIV

Algorithm Behavioral Ability  
(non-priority, short/long packets)

<u>Network Configuration</u>	Traffic Spike Level		
	<u>60%</u>	<u>80%</u>	<u>90%</u>
<u>Fully Operational</u>	-	-	-
<u>Trunk Loss</u>			
Albany - Ft. Detrick	-	-	-
Andrews - Tinker	-	-	X
Ft. Detrick - Tinker	-	-	-
Ft. Detrick - Gentile	-	-	-
Gentile - Tinker	-	-	-
<u>Link Loss</u>			
Albany - Ft. Detrick	-	X	X
Albany - Andrews	-	-	-
Andrews - Hancock	-	-	-
Andrews - Ft. Detrick	-	-	X
Andrews - Tinker	-	-	X
Ft. Detrick - Gentile	-	-	-
Ft. Detrick - Tinker	-	-	-
Gentile - Tinker	-	-	-

TABLE XIV (CONT'D)

<u>Network Configuration</u>	<u>Traffic Spike Level</u>			
	<u>60%</u>	<u>70%</u>	<u>80%</u>	<u>90%</u>
<u>Node Loss</u>				
Ft. Detrick	X	X	X	X
Gentile	-	-	-	X
Tinker	-	-	-	X

## IX Conclusions/Recommendations

The purpose behind this paper was to determine the adaptability characteristics of the hierarchical routing algorithm developed by System Control, Incorporated for the AUTODIN II computer communications network. A computer simulation program was written to model the network with the routing algorithm written in Fortran and incorporated into the simulation program.

Two network parameters were varied, traffic volume through the network and network topology in an effort to cause the routing algorithm to adapt to these changing network conditions. By taking picture "snaps" of the network status every second after a traffic spike occurred, a general idea was formed as to how the algorithm was adapting. The output statistics obtained during these "snaps" were consolidated in the form of graphs and tables in Appendices F through I.

The previous chapter presented a summary of these graphs using accepted statistical procedures in order to help the reader form some opinion on the adaptability of the routing algorithm.

Having completed the simulation exercise the author has formed these conclusions:

(1) under moderate load and no loss of primary facilities, the algorithm adapts very well and does not cause undue packet delay.

(2) the occurrence of traffic spikes and network top-



ology changes has little or no effect on the delay incurred by priority packets regardless of their size.

(3) under moderate load and no loss of primary facilities, the disturbance period is from four to five seconds.

(4) the algorithm is well-behaved during the majority of simulation runs, but in some cases, does cause packets to incur delays of up to (and exceeding), six and one half seconds.

### Recommendations

Throughout the simulation exercise there were two links that were always first to get congested. The Albany - Andrews link and the Andrews - Hancock link only contain one trunk each. It is this fact plus the amount of traffic volume they carry that causes the link to become congested. It is recommended that one additional trunk be allocated to those two links.

If the additional software overhead would not be too costly, one change is recommended in the area of the algorithm that builds the trunking routing table, TTABLE. In addition to evenly distributing the number of packets queued for output on multi-trunk links, it is recommended that the distribution of packets based on packet length also be used. Since time of transmission is directly proportional to the length of a packet, this additional qualification on packet distribution to the trunks could possibly reduce packet wait-  
time in output queues and thus reduce the overall delay

time.

The last recommendation is one of individual implementation of the algorithm subroutines into the simulation model. Instead of having 64 subroutines (eight copies of the algorithm) to represent the algorithm at the eight nodes, the implementation should be changed to have only one copy of the algorithm but accessible by the eight nodes. If this could be accomplished, then when additional nodes were added to the network, they too could access the single algorithm copy. Otherwise for each new node added to the network, eight subroutines would have to be written for that node's copy of the hierarchical routing algorithm.

### Bibliography

1. Gordon, Sidney H.. "AUTODIN II System Overview", National Telecommunications Conference, 1977.
2. Forsdick, Harry C., Richard E. Schantz and Robert H. Thomas. "Operating Systems for Computer Networks", Computer, Vol. 11, No. 1:48 (January 1978).
3. Stathopoulos, Anthony and Harold F. Caley. "The AUTODIN II Network", EASCON-77 RECORD, 5 September, 1977.
4. Western Union Telegraph Company, AUTODIN II - Network Routing, Defense Communications Agency Contract No. 200-C-637, Government Systems Division, McLean, Virginia, November 1978.
5. Schwartz, Mischa. Computer - Communications Network Design and Analysis, Prentiss - Hall Inc., Englewood, New Jersey, 1977.
6. McQuillan, John M. "Routing Algorithms for Computer Networks -- A Survey", National Telecommunications Conference, (28:1-1), 1977.
7. Systems Control, Inc. AUTODIN II Network Routing - Hierarchical Routing Design, Contract No. WDL-SC-606 280-EP, Annex 3, Palo Alto, CA., July 1978.
8. Systems Control, Inc. AUTODIN II Network Routing - Final Technical Report, Contract No. WDL-SC-606040-EC, Annex 1, Palo Alto, CA., September, 1977.
9. Bobillier, P. A., B. C. Kahan and A. R. Probst. Simulation With GPSS and GPSS V, Prentiss - Hall, Inc., Englewood Cliffs, New Jersey, 1976.
10. Schriber, Thomas J. Simulation Using GPSS, John Wiley & Sons, New York, 1974.
11. Gordon, G. The Application of GPSS V to Discrete System Simulation. Prentiss - Hall, Inc., Englewood Cliffs, New Jersey, 1975.
12. Maisel, H. and Guiliano Gnugnoli. Simulation of Discrete Stochastic Systems. Science Research Associates, Inc., Chicago, Illinois, 1972.
13. GPSSV/6000 General Information Manual. Control Data Corporation, Minneapolis, Minnesota, 1976.
14. General Purpose Simulation System/360 OS Version 2

Bibliography Cont'd

User's Manual. Program Number 5734-XS1, International Business Machines Corporation, White Plains, New York, 1969.

15. General Purpose Simulation System/360 OS (360-CS-17X) Operator's Manual. International Business Machines Corporation, White Plains, New York, 1968.
16. Stashevsky, S. and I. Adiri. "Design of a Computerized Message - Switching Center", Operations Research, Vol 26, No. 6, November - December 1978.
17. Western Union Telegraph Company. AUTODIN II/Phase I, Defense Communications Agency Contract No. 200-C-637, Part V Design (Technical) Specification - Subpart I Network Design, Government Systems Division, McLean, Virginia, April 1976.
18. Systems Control, Inc. AUTODIN II Network Routing - Update of Network Performance Analysis, Contract No. WDL-SC-606280-EP, Annex 2, Palo Alto, CA., May, 1978.
19. Defense Communications Agency. System Performance Specification (Type "A") for AUTODIN II Phase I, November, 1975.

## Appendix A

Baseline Simulation Code  
for the Andrews Node

EXPON FUNCTION RN1024 FOR POISSON INTERARRIVAL TIMES  
 1.1017.2.222/3.355/4.509/5.615/6.71.2/7.1.2/7.5.1.38  
 0.1.0/1.1.01/2.0.2.12/3.0.2.5/4.0.2.52/5.0.2.81/6.0.2.99/7.0.0.3.2  
 0.1.3.1/4.0.1.7.0.9/5.0.1.3/6.0.1.2/7.0.0.0.0

SOURCE NODE - ANDREWS  
 DESTINATIONS AND THEIR IDENTIFICATION NUMBERS  
 ALBANY - 1 HANCOCK - 5  
 ANDREWS - 2 MCLELLAN - 6  
 FIDELITY - 3 MORTON - 7  
 GENTILE - 4 TINKER - 8

DEST12 FJNC.104 AN2108 DESTINATION DISTRIBUTION AT NODE ANDREWS  
 1.119/2.1.7463/3.2.6893/4.5.7777/5.6.8355/6.7.8930/7.1.0.8

ROUTING FUNCTION FOR ANDREWS -- ROUTES PACKETS TO A PARTICULAR LINK

ROUT12 FUNCTION PH1118 ROUTING AT NODE ANDREWS  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111

PR121 FUNCTION PH2112 PRIORITY ROUTES FOR PROCESSING TIMES (SCM 1) (FIRST)  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111  
 PR122 FUNCTION PH2112 PRIORITY ROUTES FOR PROCESSING TIMES (SCM 2) (SECOND)

ANF22 FUNCTION PH3112 LENGTH ROUTES FOR XMISSION TIMES (SCM 1) (FIRST)  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111  
 ANF21 FUNCTION PH3112 LENGTH ROUTES FOR XMISSION TIMES (SCM 1) (FIRST)  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111  
 ANF11 FUNCTION PH3112 LENGTH ROUTES FOR XMISSION TIMES (SCM 1) (FIRST)  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111  
 ANF13 FUNCTION PH3112 LENGTH ROUTES FOR XMISSION TIMES (SCM 2) (SECOND)  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111  
 ANF22 FUNCTION PH3112 LENGTH ROUTES FOR XMISSION TIMES (SCM 2) (SECOND)  
 1.1.1111/2.1.1111/3.1.1111/4.1.1111/5.1.1111/6.1.1111/7.1.1111/8.1.1111  
 ANF21 FUNCTION PH3112 LENGTH ROUTES FOR XMISSION TIMES (SCM 2) (SECOND)



\* SCH PROCESSOR #1  
 ANQ11 QUEUE  
 ANPS1  
 ANQ11  
 DEPART  
 TRANSFER FN,PK121  
 LOP21 ADVANCE 2  
 TRANSFER ,CON23  
 H1P21 ADVANCE 7  
 CON23 RELEASE  
 ANPS1  
 SAVEVALUE 8,0,ANQ11,XL  
 SAVEVALUE 9,0,ANQ21,XL  
 SAVEVALUE 10,0,ANQ22,XL  
 SAVEVALUE 11,0,ANQ31,XL  
 SAVEVALUE 12,0,ANQ41,XL  
 SAVEVALUE 13,0,ANQ42,XL  
 TRANSFER FN,ROUT2

ENTER INPUT QUEUE.  
 SEIZE THE PROCESSOR FOR SERVICE.  
 LEAVE INPUT QUEUE.  
 XFER DEPENDING ON PRIORITY OF PACKET.  
 ADD 1 MSEC FOR NON-PRIORITY PROCESSING.  
 JUMP TO CONTINUE.  
 ADD 3 MSEC FOR PRIORITY PROCESSING.  
 RELEASE THE PROCESSOR.  
 STORE LENGTH OF ALBANY TRUNK1, LINK1.  
 STORE LENGTH OF DETRICK TRUNK1, LINK2.  
 STORE LENGTH OF DETRICK TRUNK2, LINK2.  
 STORE LENGTH OF HANCOCK TRUNK1, LINK3.  
 STORE LENGTH OF TINKER TRUNK1, LINK4.  
 STORE LENGTH OF TINKER TRUNK2, LINK4.  
 DEPENDING ON DESTIN, XFER TO CORRECT LINK.

\* SCH PROCESSOR #2  
 ANQ12 QUEUE  
 ANPS2  
 ANQ12  
 DEPART  
 TRANSFER FN,PK122  
 LOP22 ADVANCE 5  
 TRANSFER ,CON2A  
 H1P22 ADVANCE 3  
 CON2A RELEASE  
 ANPS2  
 SAVEVALUE 8,0,ANQ11,XL  
 SAVEVALUE 9,0,ANQ21,XL  
 SAVEVALUE 10,0,ANQ22,XL  
 SAVEVALUE 11,0,ANQ31,XL  
 SAVEVALUE 12,0,ANQ41,XL  
 SAVEVALUE 13,0,ANQ42,XL  
 TRANSFER FN,ROUT2

ENTER INPUT QUEUE.  
 SEIZE THE PROCESSOR FOR SERVICE.  
 LEAVE INPUT QUEUE.  
 XFER DEPENDING ON PRIORITY OF PACKET.  
 ADD 5 MSEC FOR NON-PRIORITY PROCESSING.  
 JUMP TO CONTINUE.  
 ADD 3 MSEC FOR PRIORITY PROCESSING.  
 RELEASE THE PROCESSOR.  
 STORE LENGTH OF ALBANY TRUNK1, LINK1.  
 STORE LENGTH OF DETRICK TRUNK1, LINK2.  
 STORE LENGTH OF DETRICK TRUNK2, LINK2.  
 STORE LENGTH OF HANCOCK TRUNK1, LINK3.  
 STORE LENGTH OF TINKER TRUNK1, LINK4.  
 STORE LENGTH OF TINKER TRUNK2, LINK4.  
 DEPENDING ON DESTIN, XFER TO CORRECT LINK.

\*  
 \* LINK 1 IS MADE UP OF 1 TRUNK TO ALBANY.  
 \* TRUNK 6 IS CONNECTED TO SCH 1.  
 \* LINK 2 IS MADE UP OF 2 TRUNKS TO SCH 2.  
 \* TRUNK 2 IS CONNECTED TO SCH 2.  
 \* TRUNK 7 IS CONNECTED TO SCH 1.  
 \* LINK 3 IS MADE UP OF 1 TRUNK TO HANCOCK.  
 \* TRUNK 1 IS CONNECTED TO SCH 2.  
 \* LINK 4 IS MADE UP OF 2 TRUNKS TO TINKER.  
 \* TRUNK 4 IS CONNECTED TO SCH 1.  
 \* TRUNK 5 IS CONNECTED TO SCH 2.  
 \*



\* ANDREWS QUEUE LENGTH DECISIONS

(2) ANQ12 TEST LE QSANQ21, QIANQ22, ANK22 IF Q21 <= Q22, FALL THRU-ELSE QU LNK 2,2.  
TRANSFER ,ANK21 GO TO QUEUE ON LINK 2,1.

ANQ14 TEST LE QSANQ41, QIANQ42, ANK42 IF Q41 <= Q42, FALL THRU-ELSE QU LNK 4,2.  
TRANSFER ,ANK41 GO TO QUEUE ON LINK 4,1.

(3) TABL M2 TABULATE RTIME  
TRANSFER FN, TMFU2  
THANS TRANSFER FN, TMPS2  
THANL TRANSFER FN, TMPL2  
ANHLIS TABULATE RTIME  
TERMINATE  
ANHL TABULATE RTIME  
TERMINATE  
ANLOS TABULATE RTIME  
TERMINATE  
ANLOL TABULATE RTIME  
TERMINATE

\* OUTPUT PROCESSING FOR SCH 1.

ANK22 QUEUE ANQ22  
SEIZE ANK3  
DEPART ANQ22  
TRANSFER FN, ANF22  
ANS22 ADVANCE 27  
TRANSFER ,CON24  
ANL22 ADVANCE 100  
CON24 RELEASE ANK3  
TRANSFER ,FTDE1

ANK41 QUEUE ANQ41  
SEIZE ANK4  
DEPART ANQ41  
TRANSFER FN, ANF41  
ANS41 ADVANCE 27  
TRANSFER ,CON25  
ANL41 ADVANCE 100  
CON25 RELEASE ANK4  
TRANSFER ,TNKR2

ANK11 QUEUE ANQ11  
SEIZE ANK6  
DEPART ANQ11  
TRANSFER FN, ANF11  
ANS11 ADVANCE 27  
TRANSFER ,CON28  
ANL11 ADVANCE 100  
CON28 RELEASE ANK6  
TRANSFER ,TNKR8

ANK12 QUEUE ANQ12  
SEIZE ANK7  
DEPART ANQ12  
TRANSFER FN, ANF12  
ANS12 ADVANCE 27  
TRANSFER ,CON29  
ANL12 ADVANCE 100  
CON29 RELEASE ANK7  
TRANSFER ,TNKR9

ANK13 QUEUE ANQ13  
SEIZE ANK8  
DEPART ANQ13  
TRANSFER FN, ANF13  
ANS13 ADVANCE 27  
TRANSFER ,CON30  
ANL13 ADVANCE 100  
CON30 RELEASE ANK8  
TRANSFER ,TNKR0

ANK14 QUEUE ANQ14  
SEIZE ANK9  
DEPART ANQ14  
TRANSFER FN, ANF14  
ANS14 ADVANCE 27  
TRANSFER ,CON31  
ANL14 ADVANCE 100  
CON31 RELEASE ANK9  
TRANSFER ,TNKR1

ANK15 QUEUE ANQ15  
SEIZE ANK0  
DEPART ANQ15  
TRANSFER FN, ANF15  
ANS15 ADVANCE 27  
TRANSFER ,CON32  
ANL15 ADVANCE 100  
CON32 RELEASE ANK0  
TRANSFER ,TNKR2

ANK16 QUEUE ANQ16  
SEIZE ANK1  
DEPART ANQ16  
TRANSFER FN, ANF16  
ANS16 ADVANCE 27  
TRANSFER ,CON33  
ANL16 ADVANCE 100  
CON33 RELEASE ANK1  
TRANSFER ,TNKR3

ANK17 QUEUE ANQ17  
SEIZE ANK2  
DEPART ANQ17  
TRANSFER FN, ANF17  
ANS17 ADVANCE 27  
TRANSFER ,CON34  
ANL17 ADVANCE 100  
CON34 RELEASE ANK2  
TRANSFER ,TNKR4

ANK18 QUEUE ANQ18  
SEIZE ANK3  
DEPART ANQ18  
TRANSFER FN, ANF18  
ANS18 ADVANCE 27  
TRANSFER ,CON35  
ANL18 ADVANCE 100  
CON35 RELEASE ANK3  
TRANSFER ,TNKR5

DEPART OUTPUT QUEUE.  
 XFER DEPENDING ON PACKET LENGTH.  
 ADD 27 MSEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 110 MSEC FOR PACKET XMISSION.  
 RELEASE TRUNK 1,6.  
 JUMP TO NODE ALBANY.

ENTER OUTPUT QUEUE FOR TRUNK 3,1.  
 SEIZE TRUNK 3,1 FOR SERVICE.  
 DEPART OUTPUT QUEUE.  
 XFER DEPENDING ON PACKET LENGTH.  
 ADD 27 MSEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 110 MSEC FOR PACKET XMISSION.  
 RELEASE TRUNK 3,1.  
 JUMP TO NODE HANCOCK.

ENTER OUTPUT QUEUE FOR TRUNK 2,2.  
 SEIZE TRUNK 2,2 FOR SERVICE.  
 DEPART OUTPUT QUEUE.  
 XFER DEPENDING ON PACKET LENGTH.  
 ADD 27 MSEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 110 MSEC FOR PACKET XMISSION.  
 RELEASE TRUNK 2,2.  
 JUMP TO NODE FT. DETRICK.

ENTER OUTPUT QUEUE FOR TRUNK 4,5.  
 SEIZE TRUNK 4,5 FOR SERVICE.  
 DEPART OUTPUT QUEUE.  
 XFER DEPENDING ON PACKET LENGTH.  
 ADD 27 MSEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 110 MSEC FOR PACKET XMISSION.  
 RELEASE TRUNK 4,5.  
 JUMP TO NODE TINKER.

♦ ♦ OUTPUT PROCESSING FOR SCM 2.

DEPART ANQ11  
 TRANSFER FN,ANF11  
 ANS11 ADVANCE 27  
 TRANSFER ,CON26  
 ANL11 ADVANCE 100  
 CON25 RELEASE ANK6  
 TRANSFER ,ALAN2

ANK31 QUEUE  
 SEIZE ANQ31  
 DEPART ANK1  
 TRANSFER FN,ANSE3  
 ANS31 ADVANCE 27  
 TRANSFER ,CON27  
 ANL31 ADVANCE 100  
 CON27 RELEASE ANK1  
 TRANSFER ,HANCK

ANK21 QUEUE  
 SEIZE ANQ21  
 DEPART ANK2  
 TRANSFER FN,ANSE2  
 ANS21 ADVANCE 27  
 TRANSFER ,CON28  
 ANL21 ADVANCE 100  
 CON25 RELEASE ANK2  
 TRANSFER ,FTOE3

ANK42 QUEUE  
 SEIZE ANQ42  
 DEPART ANK5  
 TRANSFER FN,ANSE4  
 ANS42 ADVANCE 27  
 TRANSFER ,CON29  
 ANL42 ADVANCE 100  
 CON29 RELEASE ANK5  
 TRANSFER ,INKR3

⑤





```

FJALPS1-FITTK6,QIALOU1-QITK041  RESET STATS EXCEPT THESE.
1991  RUN FOR 1 SEC @ 20% SAI., THEN PRINT STATS
FJALPS1-FITTK6,QIALOU1-QITK041  RESET STATS EXCEPT THESE.
1991  RUN FOR 1 SEC @ 20% SAI., THEN PRINT STATS
      END THIS COMPLETE SIMULATION EXERCISE.

```

```

RESET
START
RESET
START
END

```

Appendix B

Sample of Hierarchical Algorithm  
Coded in FORTRAN

Example of  
Ft. Detrick's Copy of the Algorithm

AD-A080 484

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC P/O 17/2  
AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II.(U)

DEC 79 J A WHITTENTON

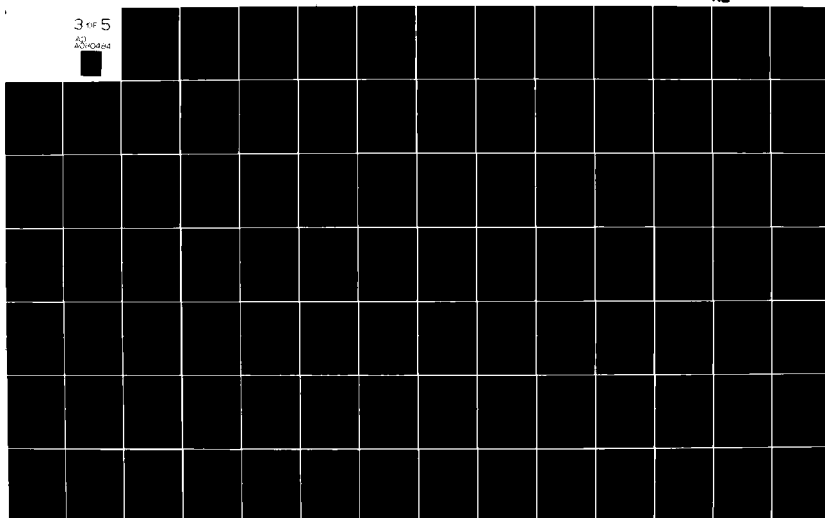
UNCLASSIFIED

AFIT/SCS/EE/79-15

NL

3 of 5

AD-A080484







```

10 DELTAS(5) = 2
   DEBFIG(1) = 1, DEBFIG(2) = 4, DEBFIG(3) = 6, DEBFIG(4) = 9
   DEBFIG(5) = 11
   DO 4) DEN1=1, DENLN
     DEB(OFN1) = DES
     DO 3) DEN2=1, DENN
       IF (DELD(OFN1, DEN2) .EQ. INFI) GO TO 4
       IF (DELD(OFN1, DEN2)) 10, 11, 20
       DEVC(OFN1) = -DEN2
       GO TO 3
     DELD(OFN1) = +DEN2
     ULS = DES + 1
     CONTINUE
     DEB(DEN1 + 1) = DES
     DESELF = 3
     UPLN = DEB(DESELF+1) - DEB(DESELF)
     DO 2) I=1, PENLN
       DEBUFF(I) = 0
       DEBUP(I) = 1
       DESTOP(I) = 1
     CONTINUE
     UPL1 = 7
     DET2 = 20
     DETOP = 1
     DO 1) I=1, 32
       DELIST(I) = 0
     CONTINUE
     DO 5) I=1, PENN
       SAVV(I/2+1) = DELD(DESELF, I)
     CONTINUE
     RETURN
     END

SUBROUTINE DEINID(PARIVA)
  INITIAL 0 INITIALIZES THE D TABLE AND BUILDS THE HOP ARRAY(N0,
  OF HOPS REQUIRED TO GET TO THE INDIVIDUAL CONNECTED NODES) AND
  THE LIST ARRAY(ARRAY OF ACTUAL CONNECTED NODE NUMBERS).
  INTEGER PARIVA(5)
  COMMON/DETAIL/DELD(4,3), DEP(9), INFI, DENN, DEVC(32), DEROUT(3),

```

```

*DEJ(J,8),DEHOP(8),DELIST(32),DESELF,DENLN,DES,DEE,DESKOT(8),
*DELI(8),DECRUFF(5),DEBUFT(12),DEBFT(5),DEPARH(6),DEJUP(5),
*DEJH,DET1,DET2,DESTJ(3)
INTEGER DES,DEMC,DEP,DEH,DLN1,DEHOP,DELIST,DEBFTIO,DEBUP,
*DESELF,DENLN,DEP,DEK,DEKOUT,DESFUL,DELLNS,DEBUFF,DEBUFT,DEPARH,
*DEJUP,DET1,DET2,DESTJ
*FAL INFI
J = DESLFF
DEJ = 1
DEK = 1
DEK = 1
DO J J L=1,DENLN
DO J J J=1,DEHN
DEJ(J,1) = 'NFI'
DEJ(J,1) = .0
CONTINUE
J = J + 1,DEHN
DEHOP(8) = JNT(INFI)
CONTINUE
DEHOP(J) =
L1 = JCP(J)
KK = DEJ(J+1) - 1
DO J J L=1,KK
DEJ(J,1) = DEHOP(1)
IF (DEH1) 81,80,70
DEJ(DEH1,DEJ) = 1.0
DEHOP(DEH1) = 1
DELI(8) (DEJ) = DEH1
DEJ = DEJ + 1
DEK = DEK + 1
CONTINUE
RETURN
END

SUBROUTINE RECOMD(PARVA)
DECO40 TAKES THE INITIAL J TABLE BUILT BY OLIND AND FILLS OUT
THE REST OF THE TABLE.

INTEGER PARVA(6)
COMMON/DETAFL/DELU(6),DEP(9),INFI,DEHN,DENC(32),DEKOUT(9),
*DEJ(3,6),DEHOP(6),DELIST(32),DESELF,DENLN,DES,DEE,DESKOT(3),

```

```

*DEL14(5),DEBUFF(5),DEBUFT(12),DEBFID(5),DEPARM(6),DEBUP(5),
*DEZ04,DET1,DET2,DESTJS(3)
*INTEGER DES,DEMC,DEP,DEPN,DEN1,DEN2,DEHOP,DELIST,DEBFID,DEBUP,
*DESELF,DENLN,DEE,DET,DEROUT,DESK01,DELINS,DEBUFF,DEBUFT,DEPARM,
*DEZ04,DET1,DET2,DESTJS
REAL INFI
1  IUES = DES - 1
   DO 20 L=DES,IOEE
   DEN1 = DELIST(L)
   DO 20 L=1,DENLN
   DO 20 J=DET,IOEE
   DEN2 = DELIST(J)
   IF (DEN2 .NE. DEN1) GO TO 13
   INTD = INT(DEU(DEN1,L))
   IF (INTD .GT. (DEHOP(DEN1) + 1)) DEU(DEN1,L) = INFI
   GO TO 23
10  DEN12 = DEU(DEN2,L) + DEU(DEN2,DEN1)
   IF (DEPN12 .LT. DEU(DEN1,L)) DEU(DEN1,L) = DEPN12
   GO TO 4
20  CONTINUE
   DET = DES
   DES = DEE
   IOES = DES - 1
   DO 20 J=DET,IOES
   DEN1 = DELIST(J)
   II = DEP(DEP1)
   KK = DEP(DEP1+1) - 1
   DO 20 J=II,KK
   DE42 = DEU(J)
   IF (DEN2 .GT. 0) GO TO 33
   GO TO 30
30  IF (DEHOP(DEN2) .GE. INFI) GO TO 45
30  IF (DEHOP(DEN2) .GT. DEHOP(DEN1)) GO TO 40
   GO TO 34
40  DO 42 L=1,DENLN
   DEPN21 = DEU(DEN1,L) + DEU(DEN1,DEN2)
   IF (DEPN21 .LT. DEU(DEN2,L)) DEU(DEN2,L) = DEPN21
   GO TO 34
42  CONTINUE
45  DELIST(DEE) = DEN2
   DEE = DEE + 1
   DEHOP(DEP2) = DEHOP(DEN1) + 1
   GO TO 35
50  CONTINUE
   IF (DEE .NE. DEE) GO TO 1

```



```

3      DEBUFF(I) = DEBUFF(I) + DEBUFT(J)
      CONTINUE
      IDEJ = IDEJ + DELINS(I)
      IDEJ = IDEJ + DELINS(I+1)
4      CONTINUE
      DO 10 L=1,DELN
      IF (DELINS(I).GT. 0) GO TO 5
      DELOCL(L) = INFI
      GO TO 10
5      DELOCL(L) = DEBUFF(L)/DELINS(L)
      CONTINUE
      DO 10 L=1,DELN
      DELOUT(L) = 0
      DESUP(L) = 0
      DESBOT(L) = INFI
      DESRES = INFI
      DO 10 L=1,DELN
          LEO = INT(LEO(N,L))
          IF ((DEI) .EQ. DEHUP(J)) .AND. (DEIN .LT. INT(INFI))) GO TO 50
          IF ((DEI) .GT. DEHUP(J)) .AND. (DELO .LT. INT(INFI))) GO TO 30
          GO TO 100
          DESRES = DELOCL(L) + JEW1*(DEO(N,L)-DEIO) + DEW2
          IF (DESRES .LT. DESRES) GO TO 40
          GO TO 100
          DESBOT(N) = L
          DESRES = DESRES
          GO TO 100
          DELEST = DELOCL(L) + JEW1*(DEO(N,L)-DEIO)
          DESRES = DELEST
          IF (DESRES .LT. DESRES) GO TO 70
          IF (DELEST .LT. DEBEST) GO TO 50
          GO TO 20
          DELOUT(N) = L
          DELEST = DELEST
          GO TO 20
          DESBOT(N) = L
          DESRES = DESRES
          GO TO 60
          CONTINUE
          RETURN
          END

```





```

C
INTEG PARVVA(6), DLO(2,1), VAR(1), FUM(2,1), SVA(1), CHA(3,1)
INTEG PARVVA(6), DLO(2,1), VAR(1), FUM(2,1), SVA(1), CHA(3,1)
INTEG PARVVA(6), DLO(2,1), VAR(1), FUM(2,1), SVA(1), CHA(3,1)
INTEG PARVVA(6), DLO(2,1), VAR(1), FUM(2,1), SVA(1), CHA(3,1)
REAL SAVVDE(1)
COMMON DETAIL/DELD(1,1), DEP(1), INFL, DENN, DLUC(32), DEROUT(8),
+D(1,1), DEUP(1), DELST(32), INSELF, DENLP, DLS, DLE, DESKO(1,1),
+DELIN(1), DEUFF(1), DEJFI(12), DEPLD(1), DEPAR(1), DEJUP(1),
+DEZONE, DET1, DET2, DESTJS(1)
INTEG DES, DEHC, DEP, DEN1, DEHC, DELIS1, DEBF10, DEBUP, DEN1,
+DESELF, DEHLI, DLE, DEKOJ, DES, DEHLIS, DEBJFF, DEBUFT, DEPARM,
+DLO(2,1), FFSTUS, DEZONE, DET1, DET2, DESTAT
REAL INFL
IF (INT(SAVVDE(14,1)) .NE. 0) GO TO 5
GO TO 11
5   DEAT = DESELF
    INDEX = DEP(OLSELF) - 1
    DEN2 = IABS(DEMC(INDEX + (INT(SAVVDE(14,7))))
    IF (INT(SAVVDE(14,2)) .GT. 1) GO TO 15
10  DESTAT = -1
    DEJUP(INFL(SAVVDE(14,7))) = 1000
    GO TO 21
15  DESTAT = 1
    DEJUP(INFL(SAVVDE(14,7))) = 1
    DELD(DEN1, DEN2) = DESTAT * ABS(DELD(DEN1, DEN2))
    I = DEP(DEN1) - 1
    I = I + 1
    DEN3 = IABS(DEHC(I))
    IF (DEN2 .NE. DEN3) GO TO 25
    DEJCO(I) = DESTAT * DEN3
    GO TO 11
25  IF (INT(SAVVDE(14,2)) .GT. 0) GO TO 116
    DELIS(INFL(SAVVDE(14,7))) = 2
    GO TO 116
116 DELIS(INFL(SAVVDE(14,7))) = 3
116 RETURN
END
C
C
C
C
C
SUBROUTINE DEMSON(PARVVA, ELU, VAK, FUM, BVA, CHA, FAC, GKO, HSA, LOS,
+QUE, STO, TAR, FMA, FSA, HMA, LMA, LMA, LMA, LMA, SAVVDE)
C

```









### Appendix C

Albany's Definitions of Subroutine Table,  
Array, Constant and Variable Names

ALLD - Link Distance (LD) table (Real), network topology matrix  
 ALD - D table (Real), contains hop/congestion information  
 ALP - P array (Integer), used as pointers in the NC array  
 ALNC - NC array (Integer), list of the connected node IDs to a particular node indexed by the entries in the ALP array  
 ALROUT - ROUTE array (Integer), identifies the link to take if Albany is used as a tandem node  
 ALSROT - SROUTE array (Integer), identifies the link to take if Albany is the source node  
 ALHOP - HOP array (Integer), number of hops required to reach each of the other nodes  
 ALLIST - LIST array (Integer), list of connected node IDs  
 ALLINS - array containing the number of trunks each link has  
 ALBUFF - array containing the number of packets queued for output on each link  
 ALBUFT - array containing the number of packets queued for output on each trunk  
 ALBFID - array containing the ID number of the first trunk in each link  
 ALLOCL - array containing the results of the calculation ALBUFF/ALLINS for each link  
 ALPARM - array containing the six parameters associated with each packet (from GPSS)  
 ALBUP - array containing the current status of each link, 1 - connected, 1000 - not connected  
 ALSTUS - array containing the previous status of each link, 1 - connected, 1000 - not connected  
 SAVVAL - array containing the SAVEVALUES passed to Fortran subroutines from GPSS  
 INFI - pseudo infinity, 1000  
 ALNN - number of nodes in network, 8  
 ALSELF - node ID for Albany, 1

ALTBUF - total number of packets queued for input processing  
 ALNLN - number of links, 4  
 ALS - variable pointing to first entry in the LIST array  
 ALE - variable pointing to last + 1 entry in LIST array  
 ALZONE - node saturation flag, 1 - saturated, 0 - not saturated  
 ALT1 - link congestion threshold, 7  
 ALT2 - node saturation threshold, 17  
 ALBNOM - congestion bias used in calculations for link selection entries in ROUTE and SROUTE arrays, 7  
 ALCNOM - extra hop bias used in similar calculations, 7  
 ALBEST - variables used in calculations to determine the entries in the ROUTE and SROUTE arrays  
 ALSBES - variable used in calculations to determine the entries in the ROUTE and SROUTE arrays  
 ALTEST - variable used in calculations to determine the entries in the ROUTE and SROUTE arrays  
 ALSTES - variables used in calculations to determine the entries in the ROUTE and SROUTE arrays  
 IDSRC - source node ID of packet (taken from parameter 4 of the packet - ALPARM (4))  
 IDSTN - destination node ID of packet (taken from parameter 1 of the packet - ALPARM (1))  
 ALNODE - index value into SAVEVALUES pointing to the first SAVEVALUE corresponding to the set of SAVEVALUES belonging to the source node of a status packet

Appendix D

Full Network Simulation Program  
Case: Ft. Detrick Goes Down

SIMULATE  
REALLOCATE BLO,3000,SV,250

SIMULATE THE AUTODIN II COMMUNICATIONS COMPUTER NETWORK

THIS IS A COMPOSITE PROGRAM WRITTEN TO SIMULATE THE AUTODIN II NETWORK WITH EMPHASIS ON THE CHARACTERISTICS OF THE HIERARCHICAL ROUTING ALGORITHM TO BE EMPLOYED IN THE REAL COMMUNICATIONS NETWORK. THE ALGORITHM IS WRITTEN USING EIGHT (8) FORTRAN SUBROUTINES. SINCE THERE WILL BE EIGHT (8) NODES SIMULATED IN THIS PROGRAM, THERE WILL BE A TOTAL OF 64 SUBROUTINES LINKED INTO THE GPSS SIMULATION MODEL. THE FIRST TWO LEVELS OF THE SUBROUTINE NAMES CORRESPOND TO THE PARTICULAR NODE, IE. AL-ALBANY, AM-ANDREWS, ETC., WITH THE LAST 4 LETTERS OF EACH SUBROUTINE NAME THE SAME IN EACH NODE, IE. ALPKR1, ALPKR2, ETC. TO REPRESENT THE PARALLEL MODULE OF THE ALGORITHM IMPLEMENTED AT EACH NODE.

THE BLOCKS ARE USED AS A COMMUNICATIONS MEDIUM AND TO PASS CONTROL FROM THE GPSS MODEL TO THE FORTRAN SUBROUTINES. FLOATING POINT SAVEVALUES ARE USED FOR TRANSFERRING TRUNK QUEUE LENGTHS FROM THE SUBROUTINES TO THE SUBROUTINES. THEY ARE ALSO USED TO HOLD THE CONTENTS OF STATUS MESSAGES THAT ARE TRANSFERRED BETWEEN MODULES. THESE SAVEVALUES ARE USED TO TRANSFER THE SELECTED TRUNK ID # BACK FROM THE ROUTING SUBROUTINE AT EACH NODE TO THE GPSS MODEL. THE SAVEVALUE IS THEN ASSIGNED TO ONE OF THE PACKETS (TRANSACTION) PARAMETERS. THIS PARAMETER IS USED IN CONJUNCTION WITH A LOGICAL FUNCTION TO TRANSFER THE PACKET ON THE SELECTED TRUNK.

Section 1 EXPON FUNCTION  $\alpha H1, C24$  FOR POISSON PACKET INTERARRIVAL TIMES

002/19,100/29,222/39,335/49,509/59,607/69,911/71,12/85,139  
003/10/31,87/88,2,12/9,2,3/32,2,52/3,2,61/25,2,59/96,3,2  
57/3,00/90,7,9/59,4,00/93,9,3/93,6,2/99,1,0/99,8,1

DESTINATIONS AND THEIR	SOURCE INDEX - ALBANY	IDENTIFICATION NUMBERS
ALBANY	1	HANCOCK
ANDOVERS	2	MCCELLAN
FOOTLOCK	3	MORTON
GENILE	4	TAYLOR

DESTIN FUNCTION RM2,00  
19-27,1/-3546,2/-7691,3/-4739,4/-3127,5/-015,6/-524,7/10,8





1194,1/.5453,2/.1893,3/.0532,+/.7777,5/.0355,6/.8930,7/1.0.8  
 \* ROUTING FUNCTION FOR ANDREWS -- ROUTES PACKETS TO A PARTICULAR LINK  
 \*  
 ROUTE FUNCTION PF3,L8 ROUTING AT NODE ANDREWS  
 1,ANL11/2,ANK21/3,ANK22/4,ANK31/5,ANK42/7,TERM2/9,UNDE2  
 \*  
 PF21 FUNCTION PF2,L2 PRIORITY ROUTES FOR PROCESSING TIMES (SCH 1)  
 1,LUP21/2,4IP21 (FIRST)  
 PRI22 FUNCTION PF2,L2 PRIORITY ROUTES FOR PROCESSING TIMES (SCH 2)  
 1,LUP22/2,4IP22 (SECOND)  
 \*  
 ANF22 FUNCTION PF3,L3 LENGTH ROUTES FOR XMISSION TIMES (SCH 1)  
 1,ANF22/2,ANL22/3,ANQ22 LENGTH ROUTES FOR XMISSION TIMES (SCH 1)  
 ANF21 FUNCTION PF3,L3 LENGTH ROUTES FOR XMISSION TIMES (SCH 1)  
 1,ANF21/2,ANL21/3,ANQ21 LENGTH ROUTES FOR XMISSION TIMES (SCH 1)  
 ANF11 FUNCTION PF3,L3 LENGTH ROUTES FOR XMISSION TIMES (SCH 1)  
 1,ANF11/2,ANL11/3,ANQ11 LENGTH ROUTES FOR XMISSION TIMES (SCH 2)  
 ANF21 FUNCTION PF3,L3 LENGTH ROUTES FOR XMISSION TIMES (SCH 2)  
 1,ANF21/2,ANL21/3,ANQ21 LENGTH ROUTES FOR XMISSION TIMES (SCH 2)  
 ANF22 FUNCTION PF3,L3 LENGTH ROUTES FOR XMISSION TIMES (SCH 2)  
 1,ANF22/2,ANL22/3,ANQ22  
 \*  
 IMF02 FUNCTION PF3,L2 LENGTH ROUTES FOR TERMINATION DECISION.  
 1,IMF02/2,IMAVL SHORT PACKETS/LONG PACKETS  
 \*  
 IMF22 FUNCTION PF2,L2 PRIORITY ROUTES FOR TERMINATION DECISIONS.  
 1,IMF22/2,IMAVS SHORT NON-PRI/SHORT PRI PACKETS  
 IMF21 FUNCTION PF2,L2 PRIORITY ROUTES FOR TERMINATION DECISIONS.  
 1,IMF21/2,IMAVL LONG NON-PRI/LONG PRI PACKETS  
 \*  
 \* SOURCE NODE - FI,DETRICK \*  
 \* DESTINATIONS AND THEIR IDENTIFICATION NUMBERS \*  
 \* ALBANY - 1 HANCOCK - 5 \*  
 \* ANDREWS - 2 MCLELLAN - 6 \*  
 \* FI,DETRICK - 3 NORFOLK - 7 \*  
 \* GENTILE - 4 TINKER - 8 \*  
 \*  
 DEST3 FUNCTION RIN2,U8 DESTINATION DISTRIBUTION AT NODE FT,DETRICK  
 0450,1/.4743,2/.1850,3/.5633,+/.0315,5/.1726,6/.7704,7/1.0.8

ROUTING FUNCTIONS FOR FT.JETTRICK -- ROUTES PACKETS TO A PARTICULAR LINK

ROUTING AT NODE FT.DELTRICK

100L11/2, DEL12/3, DEL13/4, DEL21/5, DEL22/6, DEL31/7, DEL32/8, DEL33  
 9, DEL41/10, DEL42/11, DEL51/12, DEL52/13, DEL61/14, UNDEF3

PRIORITY ROUTES FOR PROCESSING TIMES (SCH 1)  
 (FIRST)  
 (SCH 2)  
 (SECOND)  
 (SCH 3)  
 (THIRD)

LENGTH ROUTES FOR XMISSION TIMES (SCH 1)

LENGTH ROUTES FOR XMISSION TIMES (SCH 2)

LENGTH ROUTES FOR XMISSION TIMES (SCH 3)

LENGTH ROUTES FOR TERMINATION DECISION.  
 SHORT PACKETS/LONG PACKETS

PRIORITY ROUTES FOR TERMINATION DECISION.  
 SHORT NON-PRI/SHORT PRI PACKETS  
 PRIORITY ROUTES FOR TERMINATION DECISION.

ROUTING FUNCTIONS FOR FT.JETTRICK -- ROUTES PACKETS TO A PARTICULAR LINK

ROUTING AT NODE FT.DELTRICK

100L11/2, DEL12/3, DEL13/4, DEL21/5, DEL22/6, DEL31/7, DEL32/8, DEL33  
 9, DEL41/10, DEL42/11, DEL51/12, DEL52/13, DEL61/14, UNDEF3

PRIORITY ROUTES FOR PROCESSING TIMES (SCH 1)  
 (FIRST)  
 (SCH 2)  
 (SECOND)  
 (SCH 3)  
 (THIRD)

LENGTH ROUTES FOR XMISSION TIMES (SCH 1)

LENGTH ROUTES FOR XMISSION TIMES (SCH 2)

LENGTH ROUTES FOR XMISSION TIMES (SCH 3)

LENGTH ROUTES FOR TERMINATION DECISION.  
 SHORT PACKETS/LONG PACKETS

PRIORITY ROUTES FOR TERMINATION DECISION.  
 SHORT NON-PRI/SHORT PRI PACKETS  
 PRIORITY ROUTES FOR TERMINATION DECISION.

1, FY L O L / 2, FY M I L

SOURCE CODE - GENTILE	IDENTIFICATION NUMBERS
ALBANY - 1	HANDOK - 5
ANDREWS - 2	HCCLELAN - 6
FLDETROCK - 3	NORTON - 7
GENTILE - 4	TINKER - 8

DEST. FUNCTION KN2,D0 DESTINATION DISTRIBUTION AT NODE GENTILE  
0.4335,17.2333,27.0433,37.0934,47.0432,57.0376,67.0374,77.108

ROUTING FUNCTIONS FOR GLENFILE -- ROUTES PACKETS TO A PARTICULAR LINK

ROOT & FUNCTION PFS, L11 ROJING AT MODE GENTILE  
1, GEL11/2, GEL12/3, GEL13/4, GEL12/5, GEL31/6, GEL32/7, GEL41/8, GEL42/9, GEL43/10, GEL44/11, UNDEC

PRK4-1 FUNCTION 1 1, LUP-1/2, 4, P-1	PF2, L2	PRIORITY ROUTES FOR PROCESSING TIMES	(SCM 1) (FIRST)	(SCM 2) (SECOND)
PRK4-2 FUNCTION 1 1, LUP-1/2, 4, P-2	PF2, L2	" "	"	"

LENGTH ROUTES FOR XMISSION TIMES (SUM 1)

LENGTH ROUTES FOR XMISSION TIMES (SCH 2)

1,6,3,3,2,2,2,6,6,3,7,7,6,6,6,2	PF3, L3	..	..	..	..
6,5,4,1 FJHGIJUY	PF3, L3	..	..	..	..
1,6,3,4,1,2,6,6,6,1,7,3,6,6,6,1	PF3, L3	..	..	..	..
6,5,3,2 FJHGIJUY	PF3, L3	..	..	..	..
1,6,3,4,2,2,6,6,6,2,7,3,6,6,6,2	PF3, L3	..	..	..	..
6,5,1,2 FJHGIJUY	PF3, L3	..	..	..	..
1,6,3,4,2,2,6,6,6,2,7,3,6,6,6,2	PF3, L3	..	..	..	..
6,5,2,3 FJHGIJUY	PF3, L3	..	..	..	..
1,6,3,4,1,2,6,6,6,1,7,3,6,6,6,1	PF3, L3	..	..	..	..
6,5,2,3 FJHGIJUY	PF3, L3	..	..	..	..



```

*****
* SOURCE NODE - MCCLELLAN
* DESTINATIONS AND THEIR IDENTIFICATION NUMBERS
* ALJAY - 1
* ARKENS - 2
* FLETCHER - 3
* GENTLE - 4
* HAYCOCK - 5
* MCCLELLAN - 6
* MORTON - 7
* TOWER - 8
*****

DEST. FUNCTION RND,08 DESTINATION DISTRIBUTION AT NODE MCCLELLAN
1,0 01/2137,2/.0408,3/.3032,4/.3575,5/.3311,6/.7020,7/1.00,8

ROUTING FUNCTIONS FOR MCCLELLAN -- ROUTES PACKETS TO A PARTICULAR LINK

ROUTE FUNCTION PF5,L7 ROUTING AT NODE MCCLELLAN
1,MCL11/2,4L112/3,MCL21/4,M1L31/5,M1C32/6,TERM6/7,UNDEC

PRIORITY ROUTES FOR PROCESSING TIMES (SCH 1)
1,L0P11/2,M1P21 P2,L2 " " " " (FIRST)
1,L0P21/2,M1P32 P2,L2 " " " " (SCH 2)
1,L0P31/2,M1P42 P2,L2 " " " " (SECOND)

LENGTH ROUTES FOR XMISSION TIMES (SCH 1)
1,M1H31/2,M1G31/3,M0C31 " " " " "
1,M1H12/2,M1G12/3,M0C12 " " " " "
1,M1H21/2,M1G21/3,M0C21 " " " " "
1,M1H32/2,M1G32/3,M0C32 " " " " "
1,M1H11/2,M1G11/3,M0C11 " " " " "

LENGTH ROUTES FOR XMISSION TIMES (SCH 2)
1,M1H31/2,M1G31/3,M0C31 " " " " "
1,M1H12/2,M1G12/3,M0C12 " " " " "
1,M1H21/2,M1G21/3,M0C21 " " " " "
1,M1H32/2,M1G32/3,M0C32 " " " " "
1,M1H11/2,M1G11/3,M0C11 " " " " "

LENGTH ROUTES FOR TERMINATION DECISION.
SHORT PACKETS/LONG PACKETS
1,M1P31/2,M1P32 PF3,L2
1,M1P31/2,M1P32 PF3,L2

PRIORITY ROUTES FOR TERMINATION DECISION.
SHORT NO-PRI/SHORT PRI PACKETS
1,M1P31/2,M1P32 PF2,L2
1,M1P31/2,M1P32 PF2,L2

LONG NO-PRI/LONG PRI PACKETS
1,M1P31/2,M1P32 PF2,L2
1,M1P31/2,M1P32 PF2,L2

```



204





CON13 RELEASE ALP51  
 SAVEVALUE 1,Q:ALQ11,XL  
 SAVEVALUE 2,Q:ALQ21,XL  
 SAVEVALUE 3,Q:ALQ22,XL  
 SAVEVALUE 4,Q:ALQ23,XL  
 SAVEVALUE 5,Q:ALQ31,XL  
 SAVEVALUE 6,Q:ALQ32,XL  
 SAVEVALUE 7,Q:ALQ41,XL  
 SAVEVALUE 123,Q:ALQJ1,XL  
 SAVEVALUE 124,Q:ALQJ2,XL  
 HELP: ALPK1,P1,P2,P3,P4,P5,P6 GO SELECT THE TRUNK.  
 ASSIGN 5,XF1  
 TRANSFER FN,ROUT1

RELEASE THE PROCESSOR.  
 STORE LENGTH OF ANDREWS TRNK1, LINK1.  
 STORE LENGTH OF DETRICK TRNK1, LINK2.  
 STORE LENGTH OF DETRICK TRNK2, LINK2.  
 STORE LENGTH OF DETRICK TRNK3, LINK2.  
 STORE LENGTH OF NORTON TRNK1, LINK3.  
 STORE LENGTH OF NORTON TRNK2, LINK3.  
 STORE LENGTH OF TINKER TRNK1, LINK4.  
 INPUT QUEUE FOR SCM 1.  
 INPUT QUEUE FOR SCM 2.  
 MOVE TRUNK ID # TO PARAMETER 5.  
 DEPENDING ON DESTIN, XFEN TO CORRECT LINK.

{E}

\* SCM PROCESOR #2

ALBN2 QUEUE ALQ2  
 SETFN ALP52  
 DEFN ALQ2  
 TRANSFER FN,FL12  
 LOP12 ADVANCE 5  
 TRANSFER \*CON13  
 HIP12 ADVANCE 3  
 CON18 RELEASE ALP52

ENTER INPUT QUEUE.  
 SEIZE THE PROCESSOR FOR SERVICE.  
 LEAVE INPUT QUEUE.  
 XFEN DEPENDS ON PRIORITY.  
 ADD 5 USED FOR LOW PRIORITY PROCESSING.  
 JUMP TO CONTINUE.  
 ADD 3 USED FOR HIGH PRIORITY PROCESSING.  
 RELEASE THE PROCESSOR.

STORE LENGTH OF ANDREWS TRNK1, LINK1.  
 STORE LENGTH OF DETRICK TRNK1, LINK2.  
 STORE LENGTH OF DETRICK TRNK2, LINK2.  
 STORE LENGTH OF DETRICK TRNK3, LINK2.  
 STORE LENGTH OF NORTON TRNK1, LINK 3.  
 STORE LENGTH OF NORTON TRNK2, LINK 3.  
 STORE LENGTH OF TINKER TRNK1, LINK 4.  
 INPUT QUEUE FOR SCM 1.  
 INPUT QUEUE FOR SCM 2.  
 ALPK1,P1,P2,P3,P4,P5,P6 GO SELECT THE TRUNK.  
 MOVE TRUNK ID # TO PARAMETER 5.  
 DEPENDS ON DESTIN, XFEN TO CORRECT LINK.

{E}

\* LINK 1 IS MADE UP OF 1 TRUNK TO ANDREWS.  
 \* TRUNK 1 IS CONNECTED TO SCM 1.  
 \* LINK 2 IS MADE UP OF 3 TRUNKS TO F1,DETRICK.  
 \* TRUNK 2 IS CONNECTED TO SCM 2.  
 \* TRUNK 3 IS CONNECTED TO SCM 1.  
 \* TRUNK 4 IS CONNECTED TO SCM 1.  
 \* LINK 3 IS MADE UP OF 2 TRUNKS TO NORTON.  
 \* TRUNK 5 IS CONNECTED TO SCM 2.



ALK32 QUEUE	ALQ32	ENTER OUTPUT QUEUE FOR TRUNK 3,7.
SEIZE	ALTK7	SEIZE TRUNK 3,7 FOR SERVICE.
DEPART	ALQ32	DEPART OUTPUT QUEUE.
TRANSFER	FN,ALF32	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON10	JUMP TO CONTINUL.
ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON10	JUMP TO CONTINUL.
ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
RELEASE	ALTK7	RELEASE TRUNK 3,7.
TRANSFER	NORT	JUMP TO NODE NORTON.
. . . OUTPUT PROCESSING FOR SCH 2.		
ALK11 QUEUE	ALQ11	ENTER OUTPUT QUEUE FOR TRUNK 1,1.
SEIZE	ALTK1	SEIZE TRUNK 1,1 FOR SERVICE.
DEPART	ALQ11	DEPART OUTPUT QUEUE.
TRANSFER	FN,ALSE11	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON17	JUMP TO CONTINUL.
ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON17	JUMP TO CONTINUL.
ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
RELEASE	ALTK1	RELEASE TRUNK 1,1.
TRANSFER	ANY11	JUMP TO NODE ANDREWS.
ALK21 QUEUE	ALQ21	ENTER OUTPUT QUEUE FOR TRUNK 2,2.
SEIZE	ALTK2	SEIZE TRUNK 2,2 FOR SERVICE.
DEPART	ALQ21	DEPART OUTPUT QUEUE.
TRANSFER	FN,ALSE2	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON10	JUMP TO CONTINUL.
ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON18	JUMP TO CONTINUL.
ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
RELEASE	ALTK2	RELEASE TRUNK 2,2.
TRANSFER	FTD23	JUMP TO NODE FT. DICK.
ALK41 QUEUE	ALQ41	ENTER OUTPUT QUEUE FOR TRUNK 4,5.
SEIZE	ALTK4	SEIZE TRUNK 4,5 FOR SERVICE.
DEPART	ALQ41	DEPART OUTPUT QUEUE.
TRANSFER	FN,ALSE4	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON19	JUMP TO CONTINUL.
ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.

```

TRANSFER ,CON19
ALL41 ADVANCE 100
CON19 RELEASE ALIK9
ALP41 TRANSFER ,TNK43

ALP31 OUTJE ALQ31
DEPART ALIK6
DEPART ALQ31
TRANSFER FN4LSE3
ALQ31 ADVANCE 10
TRANSFER ,CON1A
ALP31 ADVANCE 27
TRANSFER ,CON1A
ALL31 ADVANCE 100
CON1A RELEASE ALIK6
ALP31 TRANSFER ,RUX1

JUMP 10 CONTINUE.
ADD 100 MSEC FOR PACKET XMISSION.
RELEASE TRUNK 499.
JUMP 10 NODE 1ANKER.

ENTER OUTPUT QUEUE FOR TRUNK 396.
SET SIZE TRUNK 396 FOR SERVICE.
DEPART OUTPUT QUEUE.
XFER DEFENDING ON PACKET LENGTH.
ADD 45 MSEC FOR STATUS PACKET PROC.
JUMP 10 CONTINUE.
ADD 27 MSEC FOR PACKET XMISSION.
JUMP 10 CONTINUE.
ADD 100 MSEC FOR PACKET XMISSION.
RELEASE TRUNK 399.
JUMP 10 NODE NORTON.

*****
NODE ADDRESS PROCESSING CENTER
2 SWITCH CONTROL MODULES (SCH) POP11/76'S
1 LOGICAL TRANSMISSION LINKS
5 PHYSICAL TRANSMISSION LINES (TRUNKS)
*****

ANDRE GENERATE 20,FN4LSE3,CON19,F GEN. PACKETS @ 20% SAT. - MEAN OF 20 MSEC
ANDR TRANSFER 001,HIGH2
ASSIGN 492
PRIORITY 1
ASSIGN 291
TRANSFER ,CON21
HIGH2 PRIORITY 2
ASSIGN 292
CON21 TRANSFER 002,LONG2
ASSIGN 391
TRANSFER ,CON22
LONG2 ASSIGN 392
CON22 ASSIGN 1,FN4LSE3?
TRANSFER 004,ANDY1,ANDY2

*****
NETWORK PACKETS ENTER THE ADDRESS NODE AT POINTS

```

```

*      ANQY1 - FOR SCH #1
*      ANQY2 - FOR SCH #2
*
* SUM PROCESSOR #1
*
ANQY1 QUEUE ANQY1
SEIZE ANQY1
DEPART ANQY1
TRANSFER FH,PR121
LUP21 ADVANCE 5
TRANSFER 3
HLP21 ADVANCE 3
LUP23 RELEASE ANQY1
SAVEVALUE 8,Q,ANQY1,XL
SAVEVALUE 9,Q,ANQY1,XL
SAVEVALUE 10,Q,ANQY2,XL
SAVEVALUE 11,Q,ANQY1,XL
SAVEVALUE 12,Q,ANQY1,XL
SAVEVALUE 13,Q,ANQY2,XL
SAVEVALUE 13,Q,ANQY1,XL
SAVEVALUE 13,Q,ANQY2,XL
HELP ANQY1,PR12,P3,PR12
ASSIGN 2,XF2
TRANSFER FH,ANQY12

ENTER INPUT QUEUE.
SEIZE THE PROCESSOR FOR SERVICE.
LEAVE INPUT QUEUE.
XFER DEPENDING ON PRIORITY OF PACKET.
ADD 5 MSEC FOR LOW PRIORITY PROCESSING.
JUMP TO CONTINUE.
ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.
RELEASE THE PROCESSOR.
STORE LENGTH OF ALBANY TRUNK1, LINK1.
STORE LENGTH OF DETROIT TRUNK1, LINK2.
STORE LENGTH OF DETROIT TRUNK2, LINK2.
STORE LENGTH OF HANCOCK TRUNK1, LINK3.
STORE LENGTH OF TINKER TRUNK1, LINK4.
STORE LENGTH OF TINKER TRUNK2, LINK4.
INPUT QUEUE FOR SCH 1.
INPUT QUEUE FOR SCH 2.
TRANSFER TRUNK ID # TO PARAMETER 5.
DEPENDING ON DESTIN, XFER TO CORRECT LINK.

* SUM PROCESSOR #2
*
ANQY2 QUEUE ANQY2
SEIZE ANQY2
DEPART ANQY2
TRANSFER FH,PR122
LUP22 ADVANCE 5
TRANSFER 3
HLP22 ADVANCE 3
LUP24 RELEASE ANQY2
SAVEVALUE 8,Q,ANQY1,XL
SAVEVALUE 9,Q,ANQY1,XL
SAVEVALUE 10,Q,ANQY2,XL
SAVEVALUE 11,Q,ANQY1,XL
SAVEVALUE 12,Q,ANQY1,XL
SAVEVALUE 13,Q,ANQY2,XL
SAVEVALUE 13,Q,ANQY1,XL
SAVEVALUE 13,Q,ANQY2,XL
HELP ANQY1,PR12,P3,PR12
ASSIGN 2,XF2
TRANSFER TRUNK ID # TO PARAMETER 5.

ENTER INPUT QUEUE.
SEIZE THE PROCESSOR FOR SERVICE.
LEAVE INPUT QUEUE.
XFER DEPENDING ON PRIORITY OF PACKET.
ADD 5 MSEC FOR LOW PRIORITY PROCESSING.
JUMP TO CONTINUE.
ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.
RELEASE THE PROCESSOR.
STORE LENGTH OF ALBANY TRUNK1, LINK1.
STORE LENGTH OF DETROIT TRUNK1, LINK2.
STORE LENGTH OF DETROIT TRUNK2, LINK2.
STORE LENGTH OF HANCOCK TRUNK1, LINK3.
STORE LENGTH OF TINKER TRUNK1, LINK4.
STORE LENGTH OF TINKER TRUNK2, LINK4.
INPUT QUEUE FOR SCH 1.
INPUT QUEUE FOR SCH 2.
TRANSFER TRUNK ID # TO PARAMETER 5.
DEPENDING ON DESTIN, XFER TO CORRECT LINK.

```

```

TRANSFER FN,ROUT2      DEPENDS ON DESTN, XFER TO CORRECT LINK.

* LINK 1 IS MADE UP OF 1 TRUNK TO ALBANY.
* TRUNK 4 IS CONNECTED TO SCM 1.
* LINK 2 IS MADE UP OF 2 TRUNKS TO FT. DETRICK.
* TRUNK 7 IS CONNECTED TO SCM 2.
* TRUNK 3 IS CONNECTED TO SCM 1.
* TRUNK 5 IS CONNECTED TO SCM 1.
* LINK 3 IS MADE UP OF 1 TRUNK TO HANCOCK.
* TRUNK 1 IS CONNECTED TO SCM 2.
* LINK 6 IS MADE UP OF 2 TRUNKS TO LINER.
* TRUNK 8 IS CONNECTED TO SCM 1.
* TRUNK 9 IS CONNECTED TO SCM 2.

* TENH2 TEST LE P3,2,ANF11  TRANSFER IF STATUS PACKET.
* TABULATE XTIME  TABULATE THIS PACKET'S TOTAL DELAY.
* TRANSFER FN,IMF02  BASED ON PACKET LENGTH, TRANSFER.
* TRANSFER FN,IMPS2  BASED ON PACKET PRIORITY, TRANSFER.
* TRANSFER FN,IMPL2  BASED ON PACKET PRIORITY, TRANSFER
* ANH15 TABULATE KTHIS  TABULATE PRI-SHORT PACKET DELAY.
* TERMINATE  DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* ANH11 TABULATE KTHIL  TABULATE PRI-LONG PACKET DELAY.
* ANH10 TABULATE RTLOS  DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* ANH09 TABULATE RTLOS  DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* ANH08 TABULATE RTLOS  DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* ANH07 TABULATE RTLOS  DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* UNH02 TERMINATE  UNDELIVERABLE PACKET TERMINATION.

* OUTPUT PROCESSING FOR SCM 1.
* ANK22 QUEUE AN022  ENTER OUTPUT QUEUE FOR TRUNK 2,3.
* SEIZE ANK3  SEIZE TRUNK 2,3 FOR SERVICE.
* DEPART AN022  DEPART OUTPUT QUEUE.
* TRANSFER FN,ANF22  XFER DEPENDS ON PACKET LENGTH.
* ADVANCE 16  ADD 16 SEC FOR STATUS PACKET PROC.
* TRANSFER *CON24  JUMP TO CONTINUE.
* ADVANCE 27  ADD 27 SEC FOR PACKET XMISSION.
* TRANSFER *CON24  JUMP TO CONTINUE.
* ADVANCE 100  ADD 100 SEC FOR PACKET XMISSION.
* CON24 RELEASE ANK3  RELEASE TRUNK 2,3.
* ANH02 TRANSFER *FIDEL  JUMP TO ADDL FT. DETRICK.

* ANK41 QUEUE AN041  ENTER OUTPUT QUEUE FOR TRUNK 4,5.
* SEIZE ANK4  SEIZE TRUNK 4,5 FOR SERVICE.
* DEPART AN041  DEPART OUTPUT QUEUE.

```

TRANSFER	FN,ANF41	XFER DEPEND ON PACKET LENGTH.
ANQ41 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON20	JUMP TO CONTINUL.
ANQ41 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON20	JUMP TO CONTINUL.
ANQ41 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON20 RELEASE	ANIK5	RELEASE TRUNK 1,1.
ANQ41 TRANSFER	TRUNK2	JUMP TO NODE LINKER.
ANQ11 QUEUE	ANQ11	ENTER OUTPUT QUEUE FOR TRUNK 1,5.
SEIZE	ANIK5	SEIZE TRUNK 1,5 FOR SERVICE.
DEPART	ANQ11	DEPART OUTPUT QUEUE.
TRANSFER	FN,ANF11	XFER DEPEND ON PACKET LENGTH.
ANQ11 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON20	JUMP TO CONTINUL.
ANQ11 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON20	JUMP TO CONTINUL.
ANQ11 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON20 RELEASE	ANIK5	RELEASE TRUNK 1,6.
ANQ11 TRANSFER	ALBN2	JUMP TO NODE ALBANY.
* * * * * OUTPUT PROCESSING FOR SCM 2. * * * * *		
ANQ31 QUEUE	ANQ31	ENTER OUTPUT QUEUE FOR TRUNK 3,1.
SEIZE	ANIK1	SEIZE TRUNK 3,1 FOR SERVICE.
DEPART	ANQ31	DEPART OUTPUT QUEUE.
TRANSFER	FN,ANSE3	XFER DEPEND ON PACKET LENGTH.
ANQ31 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON27	JUMP TO CONTINUL.
ANQ31 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON27	JUMP TO CONTINUL.
ANQ31 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON27 RELEASE	ANIK1	RELEASE TRUNK 3,1.
ANQ31 TRANSFER	HANCK	JUMP TO NODE HANCOCK.
ANQ21 QUEUE	ANQ21	ENTER OUTPUT QUEUE FOR TRUNK 2,2.
SEIZE	ANIK2	SEIZE TRUNK 2,2 FOR SERVICE.
DEPART	ANQ21	DEPART OUTPUT QUEUE.
TRANSFER	FN,ANSE2	XFER DEPEND ON PACKET LENGTH.
ANQ21 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON20	JUMP TO CONTINUL.
ANQ21 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON20	JUMP TO CONTINUL.
ANQ21 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON20 RELEASE	ANIK2	RELEASE TRUNK 2,2.





# FTOL3 - FOR SCH #3.

## SCH PROCESSOR #1

```

FTUE1 QUEUE DEQU1
SEIZE THE PROCESSOR FOR SERVICE.
DEPART INPUT QUEUE.
TRANSFER FN,PR131
LOP31 ADVANCE 5
TRANSFER 5
HUP31 ADVANCE 3
GOUN33
GOUN33 RELEASE THE PROCESSOR.
SAVEVALUE 14,0,DEQ11,XL
SAVEVALUE 15,0,DEQ12,XL
SAVEVALUE 16,0,DEQ13,XL
SAVEVALUE 17,0,DEQ21,XL
SAVEVALUE 18,0,DEQ22,XL
SAVEVALUE 19,0,DEQ31,XL
SAVEVALUE 20,0,DEQ32,XL
SAVEVALUE 21,0,DEQ33,XL
SAVEVALUE 22,0,DEQ41,XL
SAVEVALUE 23,0,DEQ42,XL
SAVEVALUE 24,0,DEQ51,XL
SAVEVALUE 25,0,DEQ52,XL
SAVEVALUE 142,0,DEQJ1,XL
SAVEVALUE 143,0,DEQJ2,XL
SAVEVALUE 144,0,DEQJ3,XL
HOLD
ASSIGN
TRANSFER FN,OUT3

```

ENTER SCH 1'S INPUT QUEUE.  
 SEIZE THE PROCESSOR FOR SERVICE.  
 DEPART INPUT QUEUE.  
 XFER DEPENDING ON PRIORITY.  
 ADD 5 MSEC FOR LOW PRIORITY PROCESSING.  
 JUMP TO CONTINUE.  
 ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.  
 RELEASE THE PROCESSOR.  
 STORE LENGTH OF ALBANY TRK1, LINK1.  
 STORE LENGTH OF ALBANY TRK2, LINK1.  
 STORE LENGTH OF ALBANY TRK3, LINK1.  
 STORE LENGTH OF ANDREWS TRK1, LINK2.  
 STORE LENGTH OF ANDREWS TRK2, LINK2.  
 STORE LENGTH OF GENTILE TRK1, LINK3.  
 STORE LENGTH OF GENTILE TRK2, LINK3.  
 STORE LENGTH OF GENTILE TRK3, LINK3.  
 STORE LENGTH OF HANCOCK TRK1, LINK4.  
 STORE LENGTH OF HANCOCK TRK2, LINK4.  
 STORE LENGTH OF TINKER TRK1, LINK5.  
 STORE LENGTH OF TINKER TRK2, LINK5.  
 INPUT QUEUE FOR SCH 1.  
 INPUT QUEUE FOR SCH 2.  
 INPUT QUEUE FOR SCH 3.  
 P,PS,P GO SELECT THE TRUNK.  
 TRANSFER THE TRUNK ID # TO PARAMETER 5.  
 DEPENDING ON DESTIN, XFER TO CORRECT LINK.

## SCH PROCESSOR #2

```

FTUE2 QUEUE DEQUE2
SEIZE THE PROCESSOR FOR SERVICE.
DEPART INPUT QUEUE.
TRANSFER FN,PR132
LOP32 ADVANCE 5
TRANSFER 5
HUP32 ADVANCE 3
GOUN34
GOUN34 RELEASE THE PROCESSOR.
SAVEVALUE 14,0,DEQ11,XL
SAVEVALUE 15,0,DEQ12,XL
SAVEVALUE 16,0,DEQ13,XL
SAVEVALUE 17,0,DEQ21,XL

```

ENTER SCH 2'S INPUT QUEUE.  
 SEIZE THE PROCESSOR FOR SERVICE.  
 DEPART INPUT QUEUE.  
 XFER DEPENDING ON PRIORITY.  
 ADD 5 MSEC FOR LOW PRIORITY PROCESSING.  
 JUMP TO CONTINUE.  
 ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.  
 RELEASE THE PROCESSOR.  
 STORE LENGTH OF ALBANY TRK1, LINK1.  
 STORE LENGTH OF ALBANY TRK2, LINK1.  
 STORE LENGTH OF ALBANY TRK3, LINK1.  
 STORE LENGTH OF ANDREWS TRK1, LINK2.

• SCH PROCE, SUR RE

LINK 1 IS MADE UP OF 3 TONK TO ALPHA.Y.

```

* TRUNK 3 IS CONNECTED TO SCM 1.
* TRUNK 4 IS CONNECTED TO SCM 2.
* TRUNK 5 IS CONNECTED TO SCM 3.
* LINK 2 IS MADE UP OF 2 TRUNKS TO ANDREWS.
* TRUNK 1 IS CONNECTED TO SCM 3.
* TRUNK 2 IS CONNECTED TO SCM 1.
* LINK 3 IS MADE UP OF 3 TRUNKS TO GENAIL.
* TRUNK 4 IS CONNECTED TO SCM 1.
* TRUNK 5 IS CONNECTED TO SCM 2.
* TRUNK 6 IS CONNECTED TO SCM 3.
* LINK 4 IS MADE UP OF 2 TRUNKS TO HANCOCK.
* TRUNK 7 IS CONNECTED TO SCM 2.
* TRUNK 8 IS CONNECTED TO SCM 3.
* LINK 5 IS MADE UP OF 2 TRUNKS TO TINKER.
* TRUNK 9 IS CONNECTED TO SCM 1.
* TRUNK 10 IS CONNECTED TO SCM 2.

```

```

TERMS TEST LE P32, DENSEL TRANSFER IF STATUS PACKET.
TABULATE THIS PACKET'S TOTAL DELAY.
TRANSFER RTIME BASED ON PACKET LENGTH, TRANSFER.
TRANSFER EN,IMFUS BASED ON PACKET PRIORITY, TRANSFER.
TABULATE EN,IMPLS BASED ON PACKET PRIORITY, TRANSFER.
TRANSFER RTHIS TABULATE PRI-SHORT PACKET DELAY.
TERMINATE DELIVER PACKET LOCALLY AND LEAVE NETWORK.
TABULATE RTHIL TABULATE PRI-LONG PACKET DELAY.
TERMINATE DELIVER PACKET LOCALLY AND LEAVE NETWORK.
FILUS TABULATE RILUS TABULATE NON-PRI SHORT PACKET DELAY.
TERMINATE DELIVER PACKET LOCALLY AND LEAVE NETWORK.
FILUL TABULATE RILUL TABULATE NON-PRI LONG PACKET DELAY.
TERMINATE DELIVER PACKET LOCALLY AND LEAVE NETWORK.
UJULS TERMINATE UNDELIVERABLE PACKET TERMINATION.

```

# OUTPUT PROCESSING FOR SCM 1.

```

DEL22 QUEU: DEQ22 ENTER OUTPUT QUEUE FOR TRUNK 2+2.
SET: DTK2 SIZE TRUNK 2+2 FOR SERVICE.
DEPART DEPART OUTPUT QUEUE.
TRANSFER EN,DEF22 XFER PENDING ON PACKET LENGTH.
ADVANCE 10 ADD 10 USEC FOR STATUS PACKET PROC.
TRANSFER 20 JUMP TO CONTINUE.
ADVANCE 27 ADD 27 USEC FOR PACKET XMISSION.
TRANSFER 30 JUMP TO CONTINUE.
ADVANCE 10 ADD 10 USEC FOR PACKET XMISSION.
RELEASE DTK2 RELEASE TRUNK 2+2.
DETERMINATE JUMP TO NODE ANDREWS.

```

DEL13 QUEUE	DEQ13	ENTER OUTPUT QUEUE FOR TRUNK 1,5.
SEIZE	DTK5	SEIZE TRUNK 1,5 FOR SERVICE.
DEPART	DEQ13	DEPART OUTPUT QUEUE.
TRANSFER	FN,DEF13	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 USED FOR STATUS PACKET PROC.
TRANSFER	*CON37	JUMP TO CONTINUE.
ADVANCE	27	ADD 27 USED FOR PACKET XMISSION.
TRANSFER	*CON37	JUMP TO CONTINUE.
ADVANCE	100	ADD 100 USED FOR PACKET XMISSION.
RELEASE	DTK5	RELEASE TRUNK 1,5.
TRANSFER	*ALBN1	JUMP TO NODE ALBANY.
DEL12 QUEUE	DEQ12	ENTER OUTPUT QUEUE FOR TRUNK 1,7.
SEIZE	DTK7	SEIZE TRUNK 1,7 FOR SERVICE.
DEPART	DEQ12	DEPART OUTPUT QUEUE.
TRANSFER	FN,DEF12	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 USED FOR STATUS PACKET PROC.
TRANSFER	*CON30	JUMP TO CONTINUE.
ADVANCE	27	ADD 27 USED FOR PACKET XMISSION.
TRANSFER	*CON36	JUMP TO CONTINUE.
ADVANCE	100	ADD 100 USED FOR PACKET XMISSION.
RELEASE	DTK7	RELEASE TRUNK 1,7.
TRANSFER	*INKR1	JUMP TO NODE INKER.
DEL31 QUEUE	DEQ31	ENTER OUTPUT QUEUE FOR TRUNK 3,8.
SEIZE	DTK3	SEIZE TRUNK 3,8 FOR SERVICE.
DEPART	DEQ31	DEPART OUTPUT QUEUE.
TRANSFER	FN,DEF31	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 USED FOR STATUS PACKET PROC.
TRANSFER	*CON39	JUMP TO CONTINUE.
ADVANCE	27	ADD 27 USED FOR PACKET XMISSION.
TRANSFER	*CON39	JUMP TO CONTINUE.
ADVANCE	100	ADD 100 USED FOR PACKET XMISSION.
RELEASE	DTK3	RELEASE TRUNK 3,8.
TRANSFER	*GENT1	JUMP TO NODE GENTILE.
* OUTPUT PROCESSING FOR 304 2.		
DEL12 QUEUE	DEQ12	ENTER OUTPUT QUEUE FOR TRUNK 1,4.
SEIZE	DTK4	SEIZE TRUNK 1,4 FOR SERVICE.
DEPART	DEQ12	DEPART OUTPUT QUEUE.
TRANSFER	FN,DEF12	XFER DEPENDS ON PACKET LENGTH.
ADVANCE	16	ADD 16 USED FOR STATUS PACKET PROC.
TRANSFER	*CON3A	JUMP TO CONTINUE.

DSH12 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON3A	JUMP TO CONTINUL.
DLG12 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON3A RELEASE	DIK1	RELEASE TRUNK 100.
DEL12 TRANSFER	ALRN1	JUMP TO NODE ALBANY.
DEL11 QUEUE	DEQ1	ENTER OUTPUT QUEUE FOR TRUNK 100.
SEIZE	DIK5	SEIZE TRUNK 100 FOR SERVICE.
DEPART	DEQ1	DEPART OUTPUT QUEUE.
TRANSFER	FN,DES1	XFER DEPENDS ON PACKET LENGTH.
DLG11 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON3B	JUMP TO CONTINUL.
DSH11 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON39	JUMP TO CONTINUL.
DLG11 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON3A RELEASE	DIK1	RELEASE TRUNK 100.
DEL11 TRANSFER	TNR2	JUMP TO NODE TINKER.
DEL32 QUEUE	DEQ32	ENTER OUTPUT QUEUE FOR TRUNK 302.
SEIZE	DIK3	SEIZE TRUNK 302 FOR SERVICE.
DEPART	DEQ32	DEPART OUTPUT QUEUE.
TRANSFER	FN,DES32	XFER DEPENDS ON PACKET LENGTH.
DLG32 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON30	JUMP TO CONTINUL.
DSH32 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON30	JUMP TO CONTINUL.
DLG32 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON30 RELEASE	DIK3	RELEASE TRUNK 302.
DEL32 TRANSFER	GEN12	JUMP TO NODE GENTLE.
DEL41 QUEUE	DEQ41	ENTER OUTPUT QUEUE FOR TRUNK 401.
SEIZE	DIK11	SEIZE TRUNK 401 FOR SERVICE.
DEPART	DEQ41	DEPART OUTPUT QUEUE.
TRANSFER	FN,DES41	XFER DEPENDS ON PACKET LENGTH.
DLG41 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON30	JUMP TO CONTINUL.
DSH41 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON30	JUMP TO CONTINUL.
DLG41 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON30 RELEASE	DIK11	RELEASE TRUNK 401.
DEL41 TRANSFER	HANCK	JUMP TO NODE HANCOCK.
DEL21 QUEUE	DEQ21	ENTER OUTPUT QUEUE FOR TRUNK 201.
DEL21 QUEUE	DEQ21	ENTER OUTPUT QUEUE FOR TRUNK 201.

DEL33	QUEUE	DTK1	SEIZE TRUNK 2,1 FOR SERVICE.
DEQ21	DEPART	DEQ21	DEPART OUTPUT QUEUE.
TRANSFER	FN,DE121		XFER DEPENDS ON PACKET LENGTH.
ADVANCE	10		ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	,CON3C		JUMP TO CONTINUE.
ADVANCE	27		ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	,CON3E		JUMP TO CONTINUE.
ADVANCE	110		ADD 110 MSEC FOR PACKET XMISSION.
RELEASE	DTK1		RELEASE TRUNK 2,1.
TRANSFER	,ANDY2		JUMP TO Y2D - ANDREWS.
DEL11	QUEUE	DEQ11	ENTER OUTPUT QUEUE FOR TRUNK 1,3.
DEQ11	DEPART	DTK3	SEIZE TRUNK 1,3 FOR SERVICE.
TRANSFER	FN,DE111		DEPART OUTPUT QUEUE.
ADVANCE	10		XFER DEPENDS ON PACKET LENGTH.
TRANSFER	,CON3F		ADD 16 MSEC FOR STATUS PACKET PROC.
ADVANCE	27		JUMP TO CONTINUE.
TRANSFER	,CON3F		ADD 27 MSEC FOR PACKET XMISSION.
ADVANCE	100		JUMP TO CONTINUE.
RELEASE	DTK3		ADD 100 MSEC FOR PACKET XMISSION.
TRANSFER	,ALAN2		RELEASE TRUNK 1,3.
DEL33	QUEUE	DEQ33	JUMP TO NOD - ALBANY.
DEQ33	DEPART	DTK10	ENTER OUTPUT QUEUE FOR TRUNK 3,10.
TRANSFER	FN,DE133		SEIZE TRUNK 3,10 FOR SERVICE.
ADVANCE	10		DEPART OUTPUT QUEUE.
TRANSFER	,CON3G		XFER DEPENDS ON PACKET LENGTH.
ADVANCE	27		ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	,CON3G		JUMP TO CONTINUE.
ADVANCE	100		ADD 27 MSEC FOR PACKET XMISSION.
RELEASE	DTK10		JUMP TO CONTINUE.
TRANSFER	,GEN12		ADD 100 MSEC FOR PACKET XMISSION.
DEL42	QUEUE	DEQ42	RELEASE TRUNK 3,10.
DEQ42	DEPART	DTK12	JUMP TO NOD - GENTILE.
TRANSFER	FN,DE142		ENTER OUTPUT QUEUE FOR TRUNK 4,12.
ADVANCE	10		SEIZE TRUNK 4,12 FOR SERVICE.
TRANSFER	,CON3H		DEPART OUTPUT QUEUE.
ADVANCE	27		XFER DEPENDS ON PACKET LENGTH.
TRANSFER	,CON3H		ADD 16 MSEC FOR STATUS PACKET PROC.
ADVANCE	110		JUMP TO CONTINUE.
RELEASE	DTK12		ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	,CON3H		JUMP TO CONTINUE.
DEL33	RELEASE	DTK10	ADD 110 MSEC FOR PACKET XMISSION.
TRANSFER			RELEASE TRUNK 4,12.





```

SAVEVALUE 20,01GE013,XL SIDE LENGTH OF DETRICK TANK3, LINK1.
SAVEVALUE 29,04GE021,XL SIDE LENGTH OF HANCOCK TANK1, LINK2.
SAVEVALUE 30,01GE011,XL SIDE LENGTH OF MCCLELLAN TANK1, LINK3.
SAVEVALUE 31,01GE032,XL SIDE LENGTH OF MCCLELLAN TANK2, LINK3.
SAVEVALUE 32,04GE041,XL SIDE LENGTH OF NORTON TANK1, LINK4.
SAVEVALUE 33,03GE031,XL SIDE LENGTH OF TINKER TANK1, LINK5.
SAVEVALUE 34,02GE032,XL SIDE LENGTH OF TINKER TANK2, LINK5.
SAVEVALUE 1-2,01GE011,XL INPUT QUEUE FOR SCH 1.
SAVEVALUE 1-3,01GE012,XL INPUT QUEUE FOR SCH 2.
SAVEVALUE HELD, P1, P2, P3, P4, P5, P6 GO SELECT THE TRUNK.
ASSIGN GEEK1, P1, P2, P3, P4, P5, P6 TRANSFER THE TANK TO # TO PARAMETER 5.
TRANSFER FN, 00014 DEPENDS ON DESIN, XFER TO CORRECT LINK.

* SUM PROCESSOR #2
GENT2 QJUF GLEU2 ENTER SCH 2'S INPUT QUEUE.
SEL23 GLEPS2 SEIZE THE PROCESSOR FOR SERVICE.
DEPART GLEU2 DEPART INPUT QUEUE.
TRANSFER FN, P142 XFER DEPENDS ON PRIORITY.
LOP42 ADVANCE 3 ACC USED FOR LOW PRIORITY PROCESSING.
TRANSFER 3 JUMP TO CONTINUE.
HOP42 ADVANCE 3 ACC USED FOR HIGH PRIORITY PROCESSING.
CONT42 GLEPS2 RELEASE THE PROCESSOR.
SAVEVALUE 20,01GE011,XL SIDE LENGTH OF DETRICK TANK1, LINK1.
SAVEVALUE 21,01GE012,XL SIDE LENGTH OF DETRICK TANK2, LINK1.
SAVEVALUE 22,01GE013,XL SIDE LENGTH OF DETRICK TANK3, LINK1.
SAVEVALUE 23,01GE021,XL SIDE LENGTH OF HANCOCK TANK1, LINK2.
SAVEVALUE 24,01GE022,XL SIDE LENGTH OF MCCLELLAN TANK1, LINK3.
SAVEVALUE 25,01GE023,XL SIDE LENGTH OF MCCLELLAN TANK2, LINK3.
SAVEVALUE 26,01GE031,XL SIDE LENGTH OF NORTON TANK1, LINK4.
SAVEVALUE 27,01GE032,XL SIDE LENGTH OF TINKER TANK1, LINK5.
SAVEVALUE 28,01GE033,XL SIDE LENGTH OF TINKER TANK2, LINK5.
SAVEVALUE 1-2,01GE011,XL INPUT QUEUE FOR SCH 1.
SAVEVALUE 1-3,01GE012,XL INPUT QUEUE FOR SCH 2.
HOLD GEEK1, P1, P2, P3, P4, P5, P6 GO SELECT THE TRUNK.
ASSIGN GLEU2 TRANSFER THE TANK TO # TO PARAMETER 5.
TRANSFER FN, 00014 DEPENDS ON DESIN, XFER TO CORRECT LINK.

* LINK 1 IS MADE UP OF 3 TANKS TO DETRICK.
* TANK 1 IS CONNECTED TO SCH 1.
* TANK 2 IS CONNECTED TO SCH 2.
* TANK 3 IS CONNECTED TO SCH 2.
* LINK 2 IS MADE UP OF 1 TANK TO HANCOCK.
* TANK 4 IS CONNECTED TO SCH 2.
* LINK 3 IS MADE UP OF 2 TANKS TO MCCLELLAN.

```

```

* TRUNK 1 IS CONNECTED TO SCM 1.
* TRUNK 2 IS CONNECTED TO SCM 2.
* LINK 4 IS MADE UP OF 1 TRUNK TO NORTON.
* LINK 3 IS CONNECTED TO SCM 1.
* LINK 5 IS MADE UP OF 2 TRUNKS TO TANKER.
* TRUNK 4 IS CONNECTED TO SCM 1.
* TRUNK 5 IS CONNECTED TO SCM 2.

* LINK TEST LE P3,2,GENE1 IF STATUS PACKET, TRANSFER.
* TABULATE THIS PACKET'S TOTAL DELAY.
* BASED ON PACKET LENGTH, TRANSFER.
* BASED ON PACKET PRIORITY, TRANSFER.
* BASED ON PACKET PRIORITY, TRANSFER.
* TABULATE PER-SHORT PACKET DELAY.
* OFFER PACKET LOCALLY AND LEAVE NETWORK.
* TABULATE PRI-LONG PACKET DELAY.
* OFFER PACKET LOCALLY AND LEAVE NETWORK.
* TABULATE NON-PRI LONG PACKET DELAY.
* OFFER PACKET LOCALLY AND LEAVE NETWORK.
* TABULATE NON-PRI LONG PACKET DELAY.
* OFFER PACKET LOCALLY AND LEAVE NETWORK.
* UNDEFINABLE PACKET TERMINATION.

* OUTPUT PROCESSING FOR SCM 1.

GEL31 QUEUE GE031 ENTER OUTPUT QUEUE FOR TRUNK 3,1.
GEL31 STATE GLTK1 SEIZE TRUNK 3,1 FOR SERVICE.
GEL31 DEPART DEPART OUTPUT QUEUE.
GEL31 TRANSFER FN,GEF31 XFER DEPENDS ON PACKET LENGTH.
GEL31 ADVANCE 10 ADD 10 MSEC FOR STATUS PACKET PROC.
GEL31 TRANSFER 10 JUMP TO CONTINUE.
GSH31 ADVANCE 27 ADD 27 MSEC FOR PACKET XMISSION.
GSH31 TRANSFER 27 JUMP TO CONTINUE.
GLH31 ADVANCE 100 ADD 100 MSEC FOR PACKET XMISSION.
GLH31 RELEASE GETK1 RELEASE TRUNK 3,1.
GEN31 TRANSFER MC011 JUMP TO NODE MC011.

GEL31 QUEUE GE031 ENTER OUTPUT QUEUE FOR TRUNK 3,1.
GEL31 STATE GETK1 SEIZE TRUNK 3,1 FOR SERVICE.
GEL31 DEPART DEPART OUTPUT QUEUE.
GEL31 TRANSFER FN,GEF51 XFER DEPENDS ON PACKET LENGTH.
GEL31 ADVANCE 10 ADD 10 MSEC FOR STATUS PACKET PROC.
GSH31 ADVANCE 27 ADD 27 MSEC FOR PACKET XMISSION.
GSH31 TRANSFER 27 JUMP TO CONTINUE.
GLH31 ADVANCE 100 ADD 100 MSEC FOR PACKET XMISSION.
GLH31 RELEASE GETK1 RELEASE TRUNK 3,1.
GEN31 TRANSFER MC011 JUMP TO NODE MC011.

```

GLG21 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GO405 RELEASE	GETK4	RELEASE TRUNK 3,4.
GEN21 TRANSFER	FNKR1	JUMP TO NODE TANKER.
GLG11 QUEUE	GEQ11	ENTER OUTPUT QUEUE FOR TRUNK 1,6.
SEIZE	GEIK5	SEIZE TRUNK 1,6 FOR SERVICE.
DEPART	GEQ11	DEPART OUTPUT QUEUE.
TRANSFER	FN,GEF11	XFER DEPENDNG ON PACKET LENGTH.
GLG11 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON47	JUMP TO CONTINUE.
USH11 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON47	JUMP TO CONTINUE.
GLG11 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GO404 RELEASE	GLIK5	RELEASE TRUNK 1,6.
GLI11 TRANSFER	FTDE1	JUMP TO NODE FT.DETRICK.
GLG41 QUEUE	GEQ41	ENTER OUTPUT QUEUE FOR TRUNK 3,3.
SEIZE	GETK3	SEIZE TRUNK 3,3 FOR SERVICE.
DEPART	GEQ41	DEPART OUTPUT QUEUE.
TRANSFER	FN,GES41	XFER DEPENDNG ON PACKET LENGTH.
GLG41 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON48	JUMP TO CONTINUE.
GS41 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON48	JUMP TO CONTINUE.
GLG41 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GO404 RELEASE	GETK3	RELEASE TRUNK 3,3.
GEN41 TRANSFER	FNORT	JUMP TO NODE NORTON.
.. OUTPUT PROCESSING FOR SCH 2.		
GLG32 QUEUE	GEQ32	ENTER OUTPUT QUEUE FOR TRUNK 3,2.
SEIZE	GEIK2	SEIZE TRUNK 3,2 FOR SERVICE.
DEPART	GEQ32	DEPART OUTPUT QUEUE.
TRANSFER	FN,GES32	XFER DEPENDNG ON PACKET LENGTH.
GLG32 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	CON49	JUMP TO CONTINUE.
USH32 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CON49	JUMP TO CONTINUE.
GLG32 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GO405 RELEASE	GETK2	RELEASE TRUNK 3,2.
GEN32 TRANSFER	FNOC12	JUMP TO NODE MCCLELLAN.
GLG22 QUEUE	GEQ22	ENTER OUTPUT QUEUE FOR TRUNK 2,5.
SEIZE	GETK5	SEIZE TRUNK 2,5 FOR SERVICE.
DEPART	GEQ22	DEPART OUTPUT QUEUE.

GE002	TRANSFER	FN,GES02	XFER DEPEND ON PACKET LENGTH.
GE002	ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
GE002	TRANSFER	CON4A	JUMP TO CONTINUL.
GE002	ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
GE002	TRANSFER	CON4A	JUMP TO CONTINUL.
GE002	ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GE002	RELEASE	GLTK9	RELEASE TRUNK 2,2.
GE002	TRANSFER	THKR3	JUMP TO NODE LINKER.
GE012	QUEUE	GE012	ENTER OUTPUT QUEUE FOR TRUNK 1,7.
GE012	SEIZE	GLTK7	SEIZE TRUNK 1,7 FOR SERVICE.
GE012	DEPART	GE012	DEPART OUTPUT QUEUE.
GE012	TRANSFER	FN,SES12	XFER DEPEND ON PACKET LENGTH.
GE012	ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
GE012	TRANSFER	CON4B	JUMP TO CONTINUL.
GE012	ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
GE012	TRANSFER	CON4B	JUMP TO CONTINUL.
GE012	ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GE012	RELEASE	GLTK7	RELEASE TRUNK 1,7.
GE012	TRANSFER	FIDE2	JUMP TO NODE FIDEBACK.
GE013	QUEUE	GE013	ENTER OUTPUT QUEUE FOR TRUNK 1,9.
GE013	SEIZE	GLTK9	SEIZE TRUNK 1,9 FOR SERVICE.
GE013	DEPART	GE013	DEPART OUTPUT QUEUE.
GE013	TRANSFER	FN,SES13	XFER DEPEND ON PACKET LENGTH.
GE013	ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
GE013	TRANSFER	CON4C	JUMP TO CONTINUL.
GE013	ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
GE013	TRANSFER	CON4C	JUMP TO CONTINUL.
GE013	ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GE013	RELEASE	GLTK9	RELEASE TRUNK 1,9.
GE013	TRANSFER	FIDE3	JUMP TO NODE FIDEBACK.
GE021	QUEUE	GE021	ENTER OUTPUT QUEUE FOR TRUNK 2,3.
GE021	SEIZE	GLTK9	SEIZE TRUNK 2,3 FOR SERVICE.
GE021	DEPART	GE021	DEPART OUTPUT QUEUE.
GE021	TRANSFER	FN,SES21	XFER DEPEND ON PACKET LENGTH.
GE021	ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
GE021	TRANSFER	CON4D	JUMP TO CONTINUL.
GE021	ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
GE021	TRANSFER	CON4D	JUMP TO CONTINUL.
GE021	ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
GE021	RELEASE	GLTK9	RELEASE TRUNK 2,3.
GE021	TRANSFER	HANDK	JUMP TO NODE HANDCOCK.



```

* OUTPUT PROCESSING
*
*   TERMS TEST LE   P3,C,HAME1
*   TABULATE      KTIME
*   TRANSFER      FN,THFUD
*   INHAL TRANSFER FN,IMPSD
*   THHAL TRANSFER FN,IMPLS
*   HAME1 TABULATE KTIME
*   TERMINATE
*
*   HAME1 TABULATE KTHIL
*   TERMINATE
*
*   HALOS TABULATE RILUS
*   TERMINATE
*
*   HALOC TABULATE RTLOL
*   TERMINATE
*   UNDES TERMINATE
*
* LINK 1 IS MADE UP OF 1 TRUNK TO ANDREWS.
*
*   HAK11 QUEUE      HNO11
*   SEIZE         HNTK1
*   DEPART        HNO11
*   TRANSFER      FN,HAG11
*   ADVANCE       10
*   TRANSFER      ,CON24
*   ADVANCE       27
*   TRANSFER      ,CON54
*   ADVANCE       100
*   CONDS RELEASE HNTK1
*   HAK11 TRANSFER ,ANDY2
*
* LINK 2 IS MADE UP OF 2 TRUNKS TO FT.DETRICK.
*
*   HAK21 QUEUE      HNO21
*   SEIZE         HNTK2
*   DEPART        HNO21
*   TRANSFER      FN,HAG21
*   ADVANCE       10
*   TRANSFER      ,CON25
*   ADVANCE       27
*   TRANSFER      ,CON25
*   ADVANCE       100
*   CONDS RELEASE HNTK2
*   HAK21 TRANSFER ,FIDE3

```

TRANSFER IF A STATUS PACKET.  
 TABULATE THIS PACKET'S TOTAL DELAY.  
 BASED ON PACKET LENGTH, TRANSFER.  
 BASED ON PACKET PRIORITY, TRANSFER.  
 BASED ON PACKET PRIORITY, TRANSFER.  
 TABULATE END-SHORT PACKET DELAY.  
 DELIVER PACKET LOCALLY AND LEAVE NETWORK.  
 TABULATE END-LONG PACKET DELAY.  
 DELIVER PACKET LOCALLY AND LEAVE NETWORK.  
 TABULATE NON-PRI SHORT PACKET DELAY.  
 DELIVER PACKET LOCALLY AND LEAVE NETWORK.  
 TABULATE NON-PRI LONG PACKET DELAY.  
 DELIVER PACKET LOCALLY AND LEAVE NETWORK.  
 UNDELIVERABLE PACKET TERMINATION.

ENTER OUTPUT QUEUE FOR TRUNK 1,1.  
 SEIZE TRUNK 1,1 FOR SERVICE.  
 DEPART OUTPUT QUEUE.  
 XFER DEPENDS ON PACKET LENGTH.  
 ADD 10 USEC FOR STATUS PACKET PROC.  
 JUMP TO CONTINUE.  
 ADD 27 USEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 100 USEC FOR PACKET XMISSION.  
 RELEASE TRUNK 1,1.  
 JUMP TO MAKE ANDREWS.

ENTER OUTPUT QUEUE FOR TRUNK 2,2.  
 SEIZE TRUNK 2,2 FOR SERVICE.  
 DEPART OUTPUT QUEUE.  
 XFER DEPENDS ON PACKET LENGTH.  
 ADD 10 USEC FOR STATUS PACKET PROC.  
 JUMP TO CONTINUE.  
 ADD 27 USEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 100 USEC FOR PACKET XMISSION.  
 RELEASE TRUNK 2,2.  
 JUMP TO FT.DETRICK.



```

ASSIGN 2,2
CON01 TRANSFER 62, LONGS
      ASSIGN 3,1
      TRANSFER 3, CON02
CON02 ASSIGN 3,2
      TRANSFER 1, FN, DEST3
      TRANSFER 2, MC011, MC012
* NETWORK PACKETS ENTER THE MC011 MC012 NODE AT POINTS:
* MC011 - FOR SCH #1.
* MC012 - FOR SCH #2.
* SUM PROCESSOR #1.
* MC011 QUEUE
  MC011 MC011
  MCPS1 MCPS1
  DEPART MC011
  TRANSFER FN, PR101
LOP61 ADVANCE 2, CON03
  TRANSFER 3, CON03
HIF61 ADVANCE 3
  RELEASE MCPS1
  SAVEVALUE 39, 0, MC011, XL
  SAVEVALUE 40, 0, MC012, XL
  SAVEVALUE 41, 0, MC021, XL
  SAVEVALUE 42, 0, MC031, XL
  SAVEVALUE 43, 0, MC032, XL
  SAVEVALUE 174, 0, MC011, XL
  SAVEVALUE 173, 0, MC012, XL
  RELEASE MCPS1, PL, P2, P3, P4, P5, P
  ASSIGN 2, YEL
  TRANSFER FN, ROUT0
* SUM PROCESSOR #2.
* MC012 QUEUE
  MC012 MC012
  MCPS2 MCPS2
  DEPART MC012
  TRANSFER FN, PR102
LOP62 ADVANCE 2, CON04
  TRANSFER 3, CON04
HIF62 ADVANCE 3
  RELEASE MCPS2
  SAVEVALUE 39, 0, MC011, XL
  SAVEVALUE 40, 0, MC012, XL
  STORE LENGTH OF GENTILE TRNK1, LINK1.
  STORE LENGTH OF GENTILE TRNK2, LINK2.
  STORE LENGTH OF TINKER TRNK1, LINK3.
  STORE LENGTH OF TINKER TRNK2, LINK3.
  INPUT QUEUE FOR SCH 1.
  INPUT QUEUE FOR SCH 2.
  TRANSFER THE TRUNK TO # TO PARAMETER 5.
  DEPENDENT ON DESTIN, XFER TO CONNECT LINK.
  ENTER SCH 1'S INPUT QUEUE.
  SEIZE THE PROCESSOR FOR SERVICE.
  DEPART INPUT QUEUE.
  XFER DEPENDENT ON PRIORITY.
  ADD 5 MSEC FOR LOW PRIORITY PROCESSING.
  JUMP TO CONTINUE.
  ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.
  RELEASE THE PROCESSOR.
  STORE LENGTH OF GENTILE TRNK1, LINK1.
  STORE LENGTH OF GENTILE TRNK2, LINK2.
  STORE LENGTH OF TINKER TRNK1, LINK3.
  STORE LENGTH OF TINKER TRNK2, LINK3.
  INPUT QUEUE FOR SCH 1.
  INPUT QUEUE FOR SCH 2.
  TRANSFER THE TRUNK TO # TO PARAMETER 5.
  DEPENDENT ON DESTIN, XFER TO CONNECT LINK.
  ENTER SCH 2'S INPUT QUEUE.
  SEIZE THE PROCESSOR FOR SERVICE.
  DEPART OUTPUT QUEUE.
  XFER DEPENDENT ON PRIORITY.
  ADD 5 MSEC FOR LOW PRIORITY PROCESSING.
  JUMP TO CONTINUE.
  ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.
  RELEASE THE PROCESSOR.
  STORE LENGTH OF GENTILE TRNK1, LINK1.
  STORE LENGTH OF GENTILE TRNK2, LINK2.

```



```

SAVEVALUE 41,0,MCO21,XL STORE LENGTH OF NORTON TRUNK, LINK2.
SAVEVALUE 42,0,MCO31,XL STORE LENGTH OF LINKER TRUNK, LINK3.
SAVEVALUE 43,0,MCO32,XL STORE LENGTH OF LINKER TRUNK, LINK3.
SAVEVALUE 47,0,MCO31,XL INPUT QUEUE FOR SC1 1.
SAVEVALUE 48,0,MCO32,XL INPUT QUEUE FOR SC1 2.
HELPD MCKR1,P1,P2,P3,P4,P5 GO SELECT THE TRUNK.
ASSIGN P,XFB TRANSFER THE TRUNK ID # TO PARAMETER 5.
TRANSFER FN,ROUTE DEPENDING ON DESTN, XFER TO CORRECT LINK.

* LINK 1 IS MADE UP OF 2 TRUNKS TO GENTLE.
* TRUNK 1 IS CONNECTED TO SC1 2.
* TRUNK 2 IS CONNECTED TO SC1 1.
* LINK 2 IS MADE UP OF 1 TRUNK TO NORTON.
* TRUNK 1 IS CONNECTED TO SC1 2.
* LINK 3 IS MADE UP OF 2 TRUNKS TO LINKER.
* TRUNK 2 IS CONNECTED TO SC1 1.
* TRUNK 3 IS CONNECTED TO SC1 2.

* TERM TEST LG P3,2,MCKE1 TRANSFER IF A STATUS PACKET.
* TABULATE RTIME TABULATE THIS PACKET'S TOTAL DELAY.
* TRANSFER FN,IMFUS BASED ON PACKET LENGTH, TRANSFER.
* TABULATE FN,IMPS6 BASED ON PACKET PRIORITY, TRANSFER.
* MCKE1 TABULATE RTIME TABULATE FIRST-SHORT PACKET DELAY.
* TERMINATE KTHLS DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* MCKE1 TABULATE KTHLS TABULATE FIRST-LONG PACKET DELAY.
* TERMINATE KTHLS DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* MCKE1 TABULATE KTHLS TABULATE NON-FAT SHORT PACKET DELAY.
* TERMINATE KTHLS DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* MCKE1 TABULATE KTHLS TABULATE NON-FAT LONG PACKET DELAY.
* TERMINATE KTHLS DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* UNDER TERMINATE UNDELAIVABLE PACKET TERMINATION.

* OUTPUT PROCESSING FOR SC1 1.
MCKE1 QUEUE MCKE1 ENTER OUTPUT QUEUE FOR TRUNK 3,2.
SETTE MCKE2 SETTE TRUNK 3,2 FOR SERVICE.
DEPART MCKE1 DEPART OUTPUT QUEUE.
TRANSFER FN,MCKE31 XFER DEPENDING ON PACKET LENGTH.
MCKE1 ADVANCE 10 ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER MCKE2 JUMP TO CONTINUE.
MCKE1 ADVANCE 27 ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER MCKE2 JUMP TO CONTINUE.
MCKE1 ADVANCE 100 ADD 100 MSEC FOR PACKET XMISSION.
CONDD RELEASE TRUNK 3,2.

```











TRANSFER	FN,ROUTE	DEPENDING ON DESTN, XFER TO CORRECT LINK.
* SUM PROCESSOR #2		
TNKN2 QUEUE	TKOU2	ENTER SUM 2'S INPUT QUEUE.
SEIZE	TKPS2	SEIZE PROCESSOR FOR SERVICE.
DEPART	TKOU2	DEPART INPUT QUEUE.
TRANSFER	FN,PR102	XFER DEPENDING ON PACKET PRIORITY.
ADVANCE	3	ADD 3 MSEC FOR LOW PRIORITY PROCESSING.
TRANSFER	3	JUMP TO CONTINUE.
ADVANCE	3	ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.
RELEASE	TKPS2	RELEASE THE PROCESSOR.
SAVEVALUE	48,01TK011,XL	STORE LENGTH OF ALBANY TRNK1, LINK1.
SAVEVALUE	49,01TK021,XL	STORE LENGTH OF ANDREWS TRNK1, LINK2.
SAVEVALUE	50,01TK022,XL	STORE LENGTH OF ANDREWS TRNK2, LINK3.
SAVEVALUE	51,01TK031,XL	STORE LENGTH OF DETROIT TRNK1, LINK3.
SAVEVALUE	52,01TK032,XL	STORE LENGTH OF DETROIT TRNK2, LINK3.
SAVEVALUE	53,01TK041,XL	STORE LENGTH OF GENTILE TRNK1, LINK3.
SAVEVALUE	54,01TK042,XL	STORE LENGTH OF GENTILE TRNK2, LINK3.
SAVEVALUE	55,01TK051,XL	STORE LENGTH OF MOCELLAN TRNK1, LINK3.
SAVEVALUE	56,01TK052,XL	STORE LENGTH OF MOCELLAN TRNK2, LINK3.
SAVEVALUE	193,01TK011,XL	INPUT QUEUE FOR SUM 1.
SAVEVALUE	193,01TK022,XL	INPUT QUEUE FOR SUM 2.
SAVEVALUE	193,01TK033,XL	INPUT QUEUE FOR SUM 3.
RELEASE	TKPS2,TKPS3,TKPS4	GO SELECT THE TRUNK.
ASSIGN	3,XF3	TRANSFER THE TRUNK 10 # TO PARAMETER 3.
TRANSFER	FN,ROUTE	DEPENDING ON DESTN, XFER TO CORRECT LINK.
* SUM PROCESSOR #3		
TNKN3 QUEUE	TKOU3	ENTER SUM 3'S INPUT QUEUE.
SEIZE	TKPS3	SEIZE PROCESSOR FOR SERVICE.
DEPART	TKOU3	DEPART INPUT QUEUE.
TRANSFER	FN,PR103	XFER DEPENDING ON PACKET PRIORITY.
ADVANCE	3	ADD 3 MSEC FOR LOW PRIORITY PROCESSING.
TRANSFER	3	JUMP TO CONTINUE.
ADVANCE	3	ADD 3 MSEC FOR HIGH PRIORITY PROCESSING.
RELEASE	TKPS3	RELEASE THE PROCESSOR.
SAVEVALUE	48,01TK011,XL	STORE LENGTH OF ALBANY TRNK1, LINK1.
SAVEVALUE	49,01TK021,XL	STORE LENGTH OF ANDREWS TRNK1, LINK2.
SAVEVALUE	50,01TK022,XL	STORE LENGTH OF ANDREWS TRNK2, LINK3.
SAVEVALUE	51,01TK031,XL	STORE LENGTH OF DETROIT TRNK1, LINK3.
SAVEVALUE	52,01TK032,XL	STORE LENGTH OF DETROIT TRNK2, LINK3.
SAVEVALUE	53,01TK041,XL	STORE LENGTH OF GENTILE TRNK1, LINK3.
SAVEVALUE	54,01TK042,XL	STORE LENGTH OF GENTILE TRNK2, LINK3.

```

SAVEVALUE 55,08TK031,XL  STORE LENGTH OF MCCLELLAN TRNK1, LINKS.
SAVEVALUE 56,01TK032,XL  STORE LENGTH OF MCCLELLAN TRNK2, LINKS.
SAVEVALUE 192,0 11QJ1,XL  INPUT QUEUE FOR SC4 1.
SAVEVALUE 193,01TQJ2,XL  INPUT QUEUE FOR SC4 2.
SAVEVALUE 194,01TQJ3,XL  INPUT QUEUE FOR SC4 3.
HELPO  TPKRT,P1,P2,P3,P4,P5,P6 GO SELECT THE TRUNK.
ADDRESS 0,XFC  TRANSFER THE TRUNK TO #17 PARAMETER 5.
TRANSFER FN,ROUTE  DEPENDS ON DELIN, XFER TO CORRECT LINK.

* LINK 1 IS MADE UP OF 1 TRUNK TO ALBANY.
* TRUNK 1 IS CONNECTED TO SC4 3.
* LINK 2 IS MADE UP OF 2 TRUNKS TO ANDOVERS.
* TRUNK 2 IS CONNECTED TO SC4 3.
* TRUNK 3 IS CONNECTED TO SC4 2.
* LINK 3 IS MADE UP OF 2 TRUNKS TO FT. DETRICK.
* TRUNK 3 IS CONNECTED TO SC4 2.
* TRUNK 4 IS CONNECTED TO SC4 1.
* TRUNK 5 IS CONNECTED TO SC4 2.
* LINK 4 IS MADE UP OF 2 TRUNKS TO GENTILE.
* TRUNK 5 IS CONNECTED TO SC4 3.
* TRUNK 6 IS CONNECTED TO SC4 1.
* LINK 5 IS MADE UP OF 2 TRUNKS TO MCCLELLAN.
* TRUNK 6 IS CONNECTED TO SC4 2.
* TRUNK 7 IS CONNECTED TO SC4 1.

* ITEM3 TEST LE P3,2,TIME1 TRANSFER IF A STATUS PACKET.
* TABULATE RTIME TABULATE THIS PACKET'S TOTAL DELAY.
* TRANSFER FN,IMF06 BASED ON PACKET LENGTH, TRANSFER.
* THIS TRANSFER FN,IMPS0 BASED ON PACKET PRIORITY, TRANSFER.
* THIL TRANSFER FN,IMPL0 BASED ON PACKET PRIORITY, TRANSFER.
* TIMES TABULATE NIMIS TABULATE PRI-SHORT PACKET DELAY.
* TERMINATE NTHIL DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* TABULATE NTHIL TABULATE PRI-LONG PACKET DELAY.
* TERMINATE RIL05 DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* TABULATE RIL05 TABULATE NON-PRI SHORT PACKET DELAY.
* TERMINATE RIL0L DELIVER PACKET LOCALLY AND LEAVE NETWORK.
* TABULATE RIL0L TABULATE NON-PRI LONG PACKET DELAY.
* TERMINATE UNDER UNDELIVERABLE PACKET TERMINATION.
* OUTPUT PROCESSING FOR SC4 1.

* TKL32 QUEUE TK032 ENTER OUTPUT QUEUE FOR TRUNK 3,5.
* SEITE TK032 SEIZE TRUNK 3,5 FOR SERVICE.
* DEPART TK032 DEPART OUTPUT QUEUE.
* TRANSFER FN,TIF32 XFER DEPENDS ON PACKET LENGTH.
* TIL32 ADVANCE 10 ACD 16 USED FOR STATUS PACKET PROC.

```



TRANSFER	,CON95	JUMP TO CONTINUE.
TSH32 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	,CON95	JUMP TO CONTINUE.
TLG32 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON95 RELEASE	TKTK9	RELEASE TRUNK 3,5.
TIN32 TRANSFER	,FIDE1	JUMP TO NODE F1.DETRICK.
TKL42 QUEUE	TKQ42	ENTER OUTPUT QUEUE FOR TRUNK 4,7.
SEIZE	TKTK7	SEIZE TRUNK 4,7 FOR SERVICE.
DEPART	TKQ42	DEPART OUTPUT QUEUE.
TRANSFER	FN,11F42	XFER DEPENDING ON PACKET LENGTH.
TIC42 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	,CON96	JUMP TO CONTINUE.
TSH42 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	,CON96	JUMP TO CONTINUE.
TLG42 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON96 RELEASE	TKTK7	RELEASE TRUNK 4,7.
TIN42 TRANSFER	,GENT1	JUMP TO NODE GENTILE.
TKL52 QUEUE	TKQ52	ENTER OUTPUT QUEUE FOR TRUNK 5,9.
SEIZE	TKTK9	SEIZE TRUNK 5,9 FOR SERVICE.
DEPART	TKQ52	DEPART OUTPUT QUEUE.
TRANSFER	FN,11F52	XFER DEPENDING ON PACKET LENGTH.
TIC52 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	,CON67	JUMP TO CONTINUE.
TSH52 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	,CON67	JUMP TO CONTINUE.
TLG52 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON67 RELEASE	TKTK9	RELEASE TRUNK 5,9.
TIN52 TRANSFER	,MULL1	JUMP TO NODE MULLERAN.
* OUTPUT PROCESSING FOR SCH 2.		
TKL22 QUEUE	TKQ22	ENTER OUTPUT QUEUE FOR TRUNK 2,3.
SEIZE	TKTK3	SEIZE TRUNK 2,3 FOR SERVICE.
DEPART	TKQ22	DEPART OUTPUT QUEUE.
TRANSFER	FN,11S22	XFER DEPENDING ON PACKET LENGTH.
TIC22 ADVANCE	10	ADD 10 MSEC FOR STATUS PACKET PROC.
TRANSFER	,CON30	JUMP TO CONTINUE.
TSH22 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	,CON30	JUMP TO CONTINUE.
TLG22 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CON30 RELEASE	TKTK3	RELEASE TRUNK 2,3.
TIN22 TRANSFER	,ANDY1	JUMP TO NODE ANDREWS.

TKL31 QUEUE	TK031	ENTER OUTPUT QUEUE FOR TRUNK 3,1.
SEIZE	TKTK4	SEIZE TRUNK 3,1 FOR SERVICE.
DEPART	TK031	DEPART OUTPUT QUEUE.
TRANSFER	FN,IT131	XFER DEPENDNG ON PACKET LENGTH.
TKL31 ADVANCE	10	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CONB9	JUMP TO CONTINUE.
TKL31 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CONB9	JUMP TO CONTINUE.
TKL31 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CONB9 RELEASE	TKTK4	RELEASE TRUNK 3,1.
TKL31 TRANSFER	RTUE2	JUMP TO NODE RT.DETRACK.
TKL51 QUEUE	TK051	ENTER OUTPUT QUEUE FOR TRUNK 5,1.
SEIZE	TKTK6	SEIZE TRUNK 5,1.
DEPART	TK051	DEPART OUTPUT QUEUE.
TRANSFER	FN,IT151	XFER DEPENDNG ON PACKET LENGTH.
TKL51 ADVANCE	10	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CONB4	JUMP TO CONTINUE.
TKL51 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CONB4	JUMP TO CONTINUE.
TKL51 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CONB4 RELEASE	TKTK6	RELEASE TRUNK 5,1.
TKL51 TRANSFER	MCUL2	JUMP TO NODE MCULELLAN.
* OUTPUT PROCESSING FOR SC4 3.		
TKL11 QUEUE	TK011	ENTER OUTPUT QUEUE FOR TRUNK 1,1.
SEIZE	TKTK1	SEIZE TRUNK 1,1 FOR SERVICE.
DEPART	TK011	DEPART OUTPUT QUEUE.
TRANSFER	FN,IT111	XFER DEPENDNG ON PACKET LENGTH.
TKL11 ADVANCE	10	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CONB8	JUMP TO CONTINUE.
TKL11 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.
TRANSFER	CONB8	JUMP TO CONTINUE.
TKL11 ADVANCE	100	ADD 100 MSEC FOR PACKET XMISSION.
CONB8 RELEASE	TKTK1	RELEASE TRUNK 1,1.
TKL11 TRANSFER	ALON2	JUMP TO NODE ALBANY.
TKL21 QUEUE	TK021	ENTER OUTPUT QUEUE FOR TRUNK 2,2.
SEIZE	TKTK2	SEIZE TRUNK 2,2 FOR SERVICE.
DEPART	TK021	DEPART OUTPUT QUEUE.
TRANSFER	FN,IT121	XFER DEPENDNG ON PACKET LENGTH.
TKL21 ADVANCE	10	ADD 16 MSEC FOR STATUS PACKET PROC.
TRANSFER	CONB6	JUMP TO CONTINUE.
TKL21 ADVANCE	27	ADD 27 MSEC FOR PACKET XMISSION.

TRANSFER ,CON3C  
 100  
 CON3C ADVANCE TK1K2  
 CON3C RELEASE ,ANDY2  
 T1K21 TRANSFER  
 TKL41 QUEUE  
 SEIZE  
 DEPART  
 TRANSFER  
 1A041 ADVANCE  
 1A041 TRANSFER ,CON3C  
 TSH41 ADVANCE  
 TRANSFER ,CON3C  
 TL641 ADVANCE 100  
 CON3C RELEASE TK1K5  
 T1K41 TRANSFER ,GENT2

JUMP TO CONTINUE.  
 ADD 100 MSEC FOR PACKET XMISSION.  
 RELEASE TRUNK 2+2.  
 JUMP TO NODE ANDREWS.  
 ENTER OUTPUT QUEUE FOR TRUNK 4+5.  
 SEIZE TRUNK 4+5 FOR SERVICE.  
 DEPART OUTPUT QUEUE.  
 XFER DEPENDENT ON PACKET LENGTH.  
 ADD 10 MSEC FOR STATUS PACKET PROC.  
 JUMP TO CONTINUE.  
 ADD 20 MSEC FOR PACKET XMISSION.  
 JUMP TO CONTINUE.  
 ADD 100 MSEC FOR PACKET XMISSION.  
 RELEASE TRUNK 4+5.  
 JUMP TO NODE GENTILE.

## Section 2

QUEUE FOR PERIODIC ENTRY INTO MESSEN

FLICK - MESSEN IS ENTERED EVERY 100 MSEC TO CHECK FOR "BAD NEWS".  
 THIS INCLUDES CHECKING THREE (3) CONDITIONS:

- (1) HAVE ANY LINKS GONE DOWN?
- (2) SINCE THE LAST TIME MESSEN WAS ENTERED, WAS THE NODE UNSATURATED AND, IS THE NODE CURRENTLY SATURATED?
- (3) ARE ANY OF THE LINKS CONGESTED?

IF THE ANSWER TO (1) OR (3) IS YES, ONE OF TWO TYPES OF STATUS MESSAGES IS GENERATED  
 (1) IF THE CURRENT STATUS OF THE NODE IS UNSATURATED, A STATUS MESSAGE IS GENERATED THAT SELECTIVELY FLOODS THE LINKS THAT ARE CONGESTED OR DOWN AND ENTERS THE APPROPRIATE VALUE IN THE STATUS MESSAGE.  
 (2) IF THE CURRENT STATUS OF THE NODE IS ONE OF SATURATION, A STATUS MESSAGE IS GENERATED WHICH DECLARES ALL THE LINKS CONGESTED.

IF THE ANSWER TO CONDITION (2) ABOVE IS YES, THE STATUS OF THE NODE IS SET TO SATURATION AND A STATUS MESSAGE IS GENERATED DISPLAYING ALL THE LINKS CONGESTED.  
 IN ANY OF THE THREE CASES, A COPY OF THE STATUS MESSAGE IS SENT TO ALL THE NETWORK NODES AND A COPY IS SENT TO THE UPDATE SUPERROUTINE OF THE LOCAL NODE.

GENERATE 100,2,0,0,F GENERATE PACKET EVERY 100 MSEC FOR MESSEN.

Ⓐ T





TRANSFER .5,ANDY1,ANDY2  
 GENN2 SPLIT 1,HANC2  
 ASSIGN 1,7  
 TRANSFER .5,ANDY1,ANDY2  
 HANC2 SPLIT 1,MCLE2  
 ASSIGN 1,9  
 TRANSFER .5,ANDY1,ANDY2  
 MCLE2 SPLIT 1,HORT2  
 ASSIGN 1,6  
 TRANSFER .5,ANDY1,ANDY2  
 HORT2 SPLIT 1,TINK2  
 ASSIGN 1,7  
 TRANSFER .5,ANDY1,ANDY2  
 TINK2 ASSIGN 1,8  
 TRANSFER .5,ANDY1,ANDY2

\*\*\*\*\* MESSAGE PROCESSING AT FT. DETRICK \*\*\*\*\*

DEIME SPLIT 1,GENME  
 SAVEVALUE 141,1,XL  
 DEIM1 OJUE DEMO1  
 SET% MESP3  
 DEPART DEMQ1  
 ASSIGN 7,3  
 SAVEVALUE 142,0,XL  
 HELPO DEMSGN,P1,P2,P3,P4,P5,P6  
 RELEASE MESP3  
 TEST L XL1,9,1,DETRI  
 TERMINATE  
 LEIRI SPLIT 1,ALRY3  
 ASSIGN 1,3  
 DEMO2 DEMO2  
 SET% DEMSP  
 DEPART DEMQ2  
 HELPO DEUPDT,P1,P2,P3,P4,P5,P6  
 HELPO DEINID,P1,P2,P3,P4,P5,P6  
 HELPO DECOMD,P1,P2,P3,P4,P5,P6  
 RELEASE DEMSP  
 DEMO2 DEMQ3  
 SET% DEMS1  
 DEPART DEMQ3  
 HELPO DEMIBL,P1,P2,P3,P4,P5,P6  
 RELEASE DEMS1  
 TERMINATE  
 ALRY3 SPLIT 1,ANUM3



DEPART	GEM03
HELPD	GERTBL,P1,P2,P3,P4,P5,P6
RELEASE	GEM51
TERMINATE	
ALBY+ SPLIT	1,ANDM4
ASSIGN	1,1
TRANSFER	5,GENT1,SENT2
ANDM+ SPLIT	1,DETR4
ASSIGN	1,2
TRANSFER	5,GENT1,SENT2
DETR+ SPLIT	1,HAND4
ASSIGN	1,3
TRANSFER	5,GENT1,SENT2
HAND4 SPLIT	1,MOLE4
ASSIGN	1,0
TRANSFER	5,GENT1,SENT2
MOLE+ SPLIT	1,NORT4
ASSIGN	1,0
TRANSFER	5,GENT1,SENT2
NORT4 SPLIT	1,TINK4
ASSIGN	1,7
TRANSFER	5,GENT1,SENT2
TINK+ SPLIT	1,0
TRANSFER	5,GENT1,SENT2
* * * * *	
MESSEN PROCESSING AT HAMCOCK	
* * * * *	
HAME SPLIT	1,HICME
SAVEVALUE	101,1,XL
HAMI QUEUE	HAM01
SETC	MESPS
DEPART	HAM01
ASSIGN	7,2
SAVEVALUE	103,0,XL
HELPD	HANSON,P1,P2,P3,P4,P5,P6
RELEASE	MESPS
TEST L	XL103,1,HAM03
TERMINATE	
HAMCO SPLIT	1,ALBYD
ASSIGN	1,5
HAME1 QUEUE	HAM02
SETC	HAMSP
DEPART	HAM02
HELPD	HAUFDT,P1,P2,P3,P4,P5,P6
HELPD	MAINIO,P1,P2,P3,P4,P5,P6
HELPD	



HELPO	HACOMD,P1,P2,P3,P4,P5,PO
RELEASE	HAMSP
HAME2	HAMF3
QUEUE	HAMS1
SAVE	HAMQ3
DEPART	HANTBL,P1,P2,P3,P4,P5,PO
HELPO	HAMS1
RELEASE	
TERMINATE	
ALBYD	1,ANDWD
SPLIT	1,1
ASSIGN	HANCK
TRANSFER	1,DETR5
ANDWS	1,2
SPLIT	HANCK
ASSIGN	1,GENNS
TRANSFER	1,3
DETR5	HANCK
ASSIGN	1,MOLES
GENND	1,7
SPLIT	HANCK
ASSIGN	1,NORTS
TRANSFER	1,8
MOLES	HANCK
SPLIT	1,TINKS
ASSIGN	1,7
TRANSFER	HANCK
TINKS	1,3
ASSIGN	HANCK
TRANSFER	
* * * * * MESSRN PROCESSING AT MCLELLAN	
MOCKE	1,NCKME
SPLIT	17,1,1,XL
SAVEVALUE	MOCH1
MOCH1	MESPO
QUEUE	
SAVE	MOCH1
DEPART	4,9
ASSIGN	
SAVEVALUE	17,9,9,XL
HELPO	MONSGH,P1,P2,P3,P4,P5,PO
RELEASE	MESPO
TEST L	XL17,9,1,MOCL4
TERMINATE	
MOCLN	1,ALBYC
SPLIT	1,6
ASSIGN	MOCH2
MOCH1	
QUEUE	

SEIZE	MCHSP
DEPART	MCHQ2
HELDC	MCHP01,P1,P2,P3,P4,P5,P6
HELPA	MCHN10,P1,P2,P3,P4,P5,P6
HELPA	MCHC00,P1,P2,P3,P4,P5,P6
RELEASE	MCHSP
QUEUE	MCHQ3
SEIZE	MCHC1
DEPART	MCHQ3
HELDC	MCHN10,P1,P2,P3,P4,P5,P6
RELEASE	MCHS1
TERMINATE	
ALRYC SPLIT	1,ARDMC
ASSIGN	1,1
TRANSFER	.5,MCHL1,4COL2
SPLIT	1,DETRO
ASSIGN	1,2
TRANSFER	.5,MCHL1,4COL2
SPLIT	1,GLNHB
ASSIGN	1,3
TRANSFER	.5,MCHL1,4COL2
SPLIT	1,HANDG
ASSIGN	1,4
TRANSFER	.5,MCHL1,4COL2
SPLIT	1,NORT6
ASSIGN	1,5
TRANSFER	.5,MCHL1,4COL2
SPLIT	1,INMG
ASSIGN	1,7
TRANSFER	.5,MCHL1,4COL2
SPLIT	1,6
TRANSFER	.5,MCHL1,4COL2
* * * * *	
MESSAGE PROCESSING AT NORION	
* * * * *	
NORME SPLIT	1,INME
SAVEVALUE	101,1,XL
QUEUE	NORM1
SEIZE	MESP7
DEPART	NORM1
ASSIGN	1,7
SAVEVALUE	103,1,XL
HELDC	NOMSGN,P1,P2,P3,P4,P5,P6
RELEASE	MESP7
TEST L	YL103,1,NORT4

NORTH	TERMINATE	1,AL9Y7
	SPLIT	1,7
	ASSIGN	NOM02
NOM01	QUEUE	NOMSP
	SPLIT	NOM02
	DEPART	NOM01,P1,P2,P3,P4,P5,P6
	HELPS	NOM01,P1,P2,P3,P4,P5,P6
	HELPS	NOM01,P1,P2,P3,P4,P5,P6
	HELPS	NOM01,P1,P2,P3,P4,P5,P6
	RELEASE	NOMSP
	QUEUE	NOM03
NOM02	SPLIT	NOM01
	DEPART	NOM03
	HELPS	NOM01,P1,P2,P3,P4,P5,P6
	RELEASE	NOM01
	TERMINATE	
AL9Y7	SPLIT	1,ANDW7
	ASSIGN	1,1
	TRANSFER	NOM01
AN0W7	SPLIT	1,DETR7
	ASSIGN	1,2
	TRANSFER	NOM01
DETR7	SPLIT	1,GENH7
	ASSIGN	1,3
	TRANSFER	NOM01
GERH7	SPLIT	1,HANC7
	ASSIGN	1,4
	TRANSFER	NOM01
HANC7	SPLIT	1,MCLE7
	ASSIGN	1,5
	TRANSFER	NOM01
MCLE7	SPLIT	1,TINK7
	ASSIGN	1,6
	TRANSFER	NOM01
TINK7	ASSIGN	1,8
	TRANSFER	NOM01
	MESSAGE PROCESSING AT TINKER	
TIME	SAVEVALUE	191,1,XL
TIME	QUEUE	TAM01
	SPLIT	MSP05
	DEPART	TIM01
	ASSIGN	7,5
	SAVEVALUE	195,0,XL

HELPO	TIMSGN,P1,P2,P3,P4,P5,P6
RELEASE	MESPO
TEST L	XL195,1,TIKER
TERMINATE	
TIKER SPLIT	1,ALBY8
ASSIGN	1,8
TIME1 QUEUE	TIME2
SEIZE	TIMEP
DEPART	TIME2
HELPO	ILUPDI,P1,P2,P3,P4,P5,P6
HELPA	ILIND,P1,P2,P3,P4,P5,P6
HELPA	ILICOM,P1,P2,P3,P4,P5,P6
RELEASE	TIMEP
QUEUE	TIME3
SEIZE	TIME1
DEPART	TIME3
HELPO	ILIRBL,P1,P2,P3,P4,P5,P6
RELEASE	TIME1
TERMINATE	
ALBY8 SPLIT	1,ANDW8
ASSIGN	1,1
COMM6 TRANSFER	.J33,TNK3
TRANSFER	.D,TNK1,TNK2
ANDW3 SPLIT	1,OLINO
ASSIGN	1,2
TRANSFER	COMM6
SPLIT	1,GEN6
ASSIGN	1,3
TRANSFER	COMM6
SPLIT	1,HAND8
ASSIGN	1,4
TRANSFER	COMM6
SPLIT	1,HOLE8
ASSIGN	1,5
TRANSFER	COMM6
SPLIT	1,NORT8
ASSIGN	1,5
TRANSFER	COMM6
SPLIT	1,7
ASSIGN	1,7
TRANSFER	COMM6

\* \* \* \*

### Section 3

STICK - MESSAGE IS ENTERED EVERY 101 MSEC TO CHECK FOR "GOOD NEWS".  
THIS INCLUDES CHECKING THREE (3) CONDITIONS:

- (1) HAVE ANY LINKS COME UP?
  - (2) HAVE ANY LINKS GONE UNCONGESTED?
  - (3) SINCE THE LAST TIME MESSAGE WAS ENTERED, WAS THE NODE SATURATED AND, IS THE NODE CURRENTLY UNSATURATED?
- AS IN FICK, AN APPROPRIATE STATUS MESSAGE IS GENERATED, SPECIALLY IF ANY OF THE ANSWERS TO THE ABOVE CONDITIONS IS YES.

GENERATE 4 PACKETS EVERY 401 MSEC.  
GIVE AT PRIORITY CLASSIFICATION.  
SET PARAMETER 2 WITH THE PRIORITY.  
SET PARAMETER 3 WITH THE SPECIAL LENGTH.  
SEND COPY FOR ANDREWS PREPARATION.  
SET FLAG FOR STICK PROCESSING.  
GO THRU SAME PROCESSING AS FICK.

GENERATE 401,200,000  
PRIORITY 2  
ASSIGN 2,2  
ASSIGN 3,3  
SPLIT 1,ANDR  
SAVEVALUE 122,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 131,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 140,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 149,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 158,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 167,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 176,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 185,0,XL  
TRANSFER 1,ALPH  
SPLIT 1,ORIN  
SAVEVALUE 194,0,XL  
TRANSFER 1,ALPH

ALY  
ANDR  
DETR  
GENE  
HACK  
HOLN  
HORN  
TINK

### Section 4

CODE FOR PERIODIC ENTRY INTO TABLE

TABLE IS ENTERED PERIODICALLY EVERY 100 MSEC TO RECALCULATE

THE SOURCE AND ROUTE ARRAYS. THESE ARE ALWAYS CONTAINING THE  
LINK NO. FOR THE PACKET TO BE ROUTED OVER FOR RESPECTIVELY  
SOURCE NODE ROUTING AND TANDER4 NODE ROUTING.

```

GENERATE 100,3,,,F
SPLIT 1,ANDS
TRANSFER 1,ALME2
SPLIT 1,DEIK
TRANSFER 1,ANME2
SPLIT 1,GENI
TRANSFER 1,DEME2
SPLIT 1,HNOC
TRANSFER 1,MOLE
SPLIT 1,HAME2
TRANSFER 1,NOUW
SPLIT 1,MOHE2
TRANSFER 1,TIKR
SPLIT 1,NUME2
TRANSFER 1,TIME2

```

## Section 5

INITIALIZATION OF ALGORITHMIC TABLES, ARRAYS, CONSTANTS, AND FLAGS.  
(LD TABLES, D TABLES, NC ARRAYS, P ARRAYS, HOP ARRAYS, AND LIST ARRAYS)

GENERATE 1,,,1,,,F GENERATE ONLY 1 PACKET TO INITIALIZE THE SUMS.

ALBANY

```

HELPC ALINIT,P1,P2,P3,P4,P5,P6
HELPA ALINIO,P1,P2,P3,P4,P5,P6
HELPS ALCOHO,P1,P2,P3,P4,P5,P6
HELPC ALKIBL,P1,P2,P3,P4,P5,P6

```

ANOR:MS

```

HELPC ANINIT,P1,P2,P3,P4,P5,P6
HELPA ANINIO,P1,P2,P3,P4,P5,P6
HELPA ANOCMU,P1,P2,P3,P4,P5,P6
HELPC ANKIBL,P1,P2,P3,P4,P5,P6

```

FT.DETRICK

HELPC	DEINIT, P1, P2, P3, P4, P5, P6
HELPA	DEINID, P1, P2, P3, P4, P5, P6
HELPA	DECOMD, P1, P2, P3, P4, P5, P6
HELPC	DEIBL, P1, P2, P3, P4, P5, P6
GENTLE	
HELPC	GEINIT, P1, P2, P3, P4, P5, P6
HELPA	GEINID, P1, P2, P3, P4, P5, P6
HELPA	GECOMD, P1, P2, P3, P4, P5, P6
HELPC	GEIBL, P1, P2, P3, P4, P5, P6
HANJOCK	
HELPC	HAINIT, P1, P2, P3, P4, P5, P6
HELPA	HAINID, P1, P2, P3, P4, P5, P6
HELPA	HACOMD, P1, P2, P3, P4, P5, P6
HELPC	HARIBL, P1, P2, P3, P4, P5, P6
MCQUELLAN	
HELPC	MCINAT, P1, P2, P3, P4, P5, P6
HELPA	MCINID, P1, P2, P3, P4, P5, P6
HELPA	MCCOMD, P1, P2, P3, P4, P5, P6
HELPC	MCIBL, P1, P2, P3, P4, P5, P6
LORTON	
HELPC	NOINIT, P1, P2, P3, P4, P5, P6
HELPA	NOINID, P1, P2, P3, P4, P5, P6
HELPA	NOCOMD, P1, P2, P3, P4, P5, P6
HELPC	NOIBL, P1, P2, P3, P4, P5, P6
TINKER	
HELPC	TINIT, P1, P2, P3, P4, P5, P6
HELPA	TINID, P1, P2, P3, P4, P5, P6
HELPA	TICOMD, P1, P2, P3, P4, P5, P6
HELPC	TIBL, P1, P2, P3, P4, P5, P6
TENINATE	

CODE FOR CHANGING INTO CONNECTIVITY CHANGE

Section 6









255

```

F3ALPS1-FIT11451,Q3ALQ01-Q11M03  RESET STATS EXCEPT THESE.
1991  RUN FOR 1 SEC @ 20% SAT., THEN PRINT STATS.
F3ALPS1-FIT11451,Q3ALQ01-Q11M03  RESET STATS EXCEPT THESE.
1991  RUN FOR 1 SEC @ 20% SAT., THEN PRINT STATS.
      END TH1, COMP-LIE SIMULATION EXERCISE.

```

```

RESET
START
RESET
START
END

```

Appendix E

Listing of SAVEVALUES

# Albany

SAVEVALUE		Reason for Use
1	Length of output queue	ALQ11
2	" " " "	ALQ21
3	" " " "	ALQ22
4	" " " "	ALQ23
5	" " " "	ALQ31
6	" " " "	ALQ32
7	" " " "	ALQ41
57 - 64	Network status information (Albany's row of the LD matrix)	
121	Link change flag if non-zero, trunk change if zero	
122	MESGEN entry flag, 0 - STICK, 1 - FTICK	
123	Length of input queue	ALQU1
124	" " " "	ALQU2
125	MESGEN status message generation flag, 0 - no msg, 1 - msg generated	
126	Trunk/Link change flag, 0 - drop, 1 - add	
127	Link number in which change is to occur	

Andrews

SAVEVALUE		Reason for Use
8	Length of output queue	ANQ11
9	" " " "	ANQ21
10	" " " "	ANQ22
11	" " " "	ANQ31
12	" " " "	ANQ41
13	" " " "	ANQ42
65 - 72	Network status information (Andrews' row of LD matrix	
130	Link change flag if non-zero, trunk change flag if zero	
131	MESGEN entry flag, 0 - STICK, 1 - FTICK	
132	Length of input queue	ANQU1
133	" " " "	ANQU2
134	MESGEN status message generation flag, 0 - no msg, 1 - msg generation	
135	Trunk/Link change flag, 0 - drop, 1 - add	
136	Link number in which change is to occur	

Ft. Detrick

SAVEVALUE		Reason for Use
14	Length of output queue	DEQ11
15	" " " "	DEQ12
16	" " " "	DEQ13
17	" " " "	DEQ21
18	" " " "	DEQ22
19	" " " "	DEQ31
20	" " " "	DEQ32
21	" " " "	DEQ33
22	" " " "	DEQ41
23	" " " "	DEQ42
24	" " " "	DEQ51
25	" " " "	DEQ52
73 - 80	Network status information (Ft. Detrick's row of the LD matrix)	
140	Link change flag if non-zero, trunk change if zero	
141	MESGEN entry flag, 0 - STICK, 1 - FTICK	
142	Length of input queue	DEQU1
143	" " " "	DEQU2
144	" " " "	DEQU3
145	MESGEN status message generation flag, 0 - no msg, 1 - message generated	
146	Trunk/Link change flag, 0 - drop, 1 - add	
147	Link number in which change is to occur	



# Gentile

SAVEVALUE		Reason for Use
26	Length of output queue	GEQ11
27	" " " "	GEQ12
28	" " " "	GEQ13
29	" " " "	GEQ21
30	" " " "	GEQ31
31	" " " "	GEQ32
32	" " " "	GEQ41
33	" " " "	GEQ51
34	" " " "	GEQ52
81 - 88	Network status information (Gentile's row of the LD matrix)	
150	Link change flag if non-zero, trunk change if zero	
151	MESGEN entry flag, 0 - STICK, 1 - FTICK	
152	Length of input queue	GEQU1
153	" " " "	GEQU2
154	MESGEN status generation flag, 0 - no msg, 1 - msg generated	
155	Trunk/Link change flag, 0 - drop, 1 - add	
156	Link number in which change is to occur	

Hancock

SAVEVALUE		Reason for Use
35	Length of output queue	HNQ11
36	" " " "	HNQ21
37	" " " "	HNQ22
38	" " " "	HNQ31
89 - 96	Network status information (Hancock's row of the LD matrix)	
160	Link change flag if non-zero, trunk chnage if zero	
161	MESGEN entry flag, 0 - STICK, 1 - FTICK	
162	Length of input queue	HNQUE
163	MESGEN status message generation flag, 0 - no msg, 1 - msg generated	
164	Trunk/Link change flag, 0 - drop, 1 - add	
165	Link number in which change is to occur	

McClellan

SAVEVALUE		Reason for Use
39	Length of output queue	MCQ11
40	" " " "	MCQ12
41	" " " "	MCQ21
42	" " " "	MCQ31
43	" " " "	MCQ32
97 - 104	Network status information (McClellan's row of the LD matrix)	
170	Link change flag if non-zero, trunk change if zero	
171	MESGEN entry flag, 0 - STICK, 1 - FTICK	
172	Length of input queue	MCQU1
173	" " " "	MCQU2
174	MESGEN status message generation flag, 0 - no msg, 1 - msg generated	
175	Trunk/Link change flag, 0 - drop, 1 - add	
176	Link number in which change is to occur	

Norton

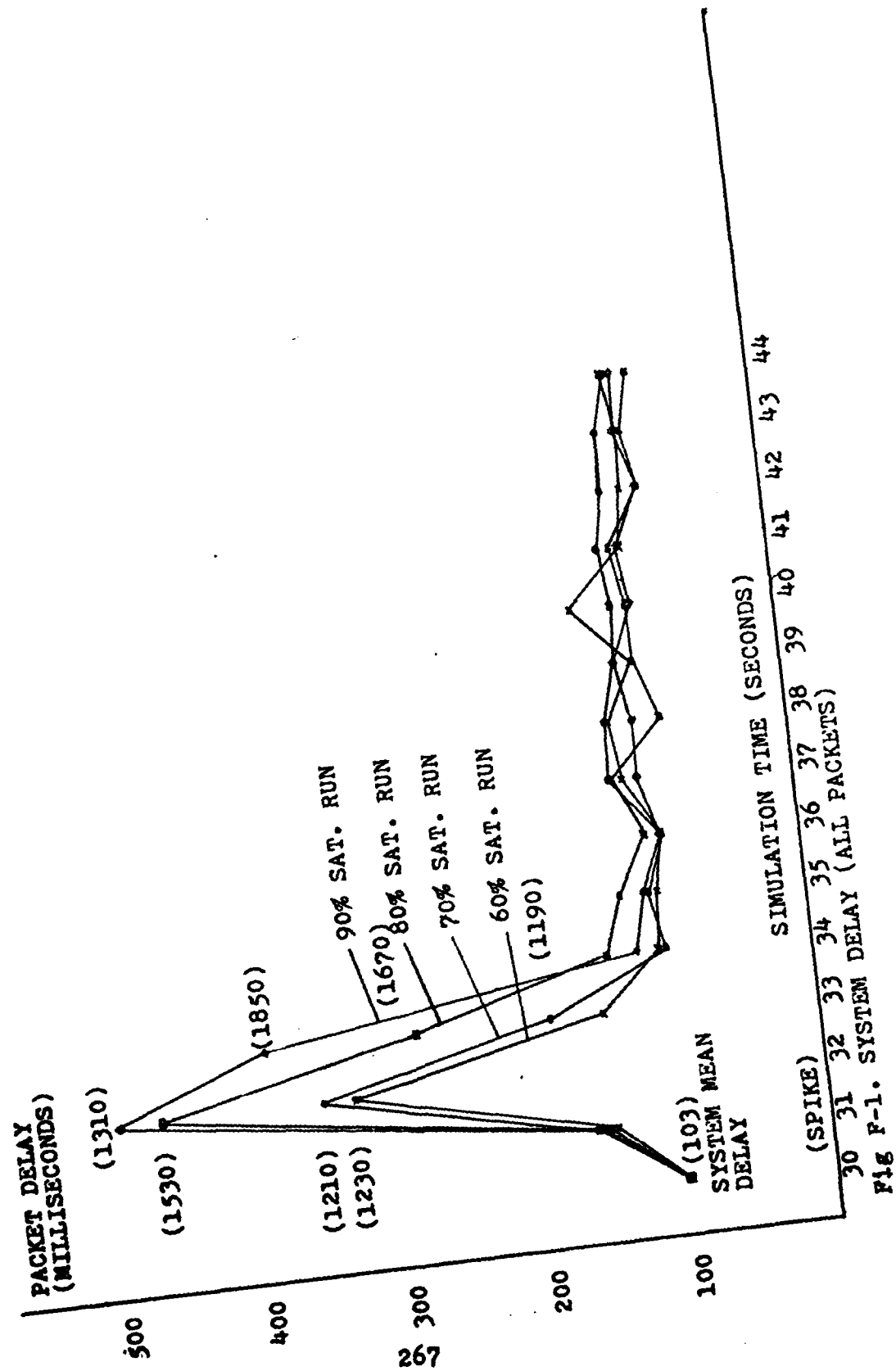
SAVEVALUE		Reason for Use
44	Length of output queue	NOQ11
45	" " " "	NOQ12
46	" " " "	NOQ21
47	" " " "	NOQ31
105 - 112	Network status message information (Norton's row of the LD matrix)	
180	Link change flag if non-zero, trunk change if zero	
181	MESGEN entry flag, 0 - STICK, 1 - FTICK	
182	Length of input queue	NOQUE
183	MESGEN status message generation flag, 0 - no msg, 1 - msg generated	
184	Trunk/Link change flag, 0 - drop, 1 - add	
185	Link number in which change is to occur	

# Tinker

SAVEVALUE		Reason for Use
48	Length of output queue	TKQ11
49	" " " "	TKQ21
50	" " " "	TKQ22
51	" " " "	TKQ31
52	" " " "	TKQ32
53	" " " "	TKQ41
54	" " " "	TKQ42
55	" " " "	TKQ51
56	" " " "	TKQ52
113 - 120	Network status information (Tinker's row of the LD matrix)	
190	Link change flag if non-zero, trunk change if zero	
191	MESGEN entry flag, 0 - STICK, 1 - FTICK	
192	Length of input queue	TKQU1
193	" " " "	TKQU2
194	" " " "	TKQU3
195	MESGEN status message generation flag, 0 - no msg, 1 - msg generated	
196	Trunk/Link change flag, 0 - drop, 1 - add	
197	Link number in which change is to occur	

Appendix F

Graphical and Tabular Delay Results  
From the Fully Operational Network



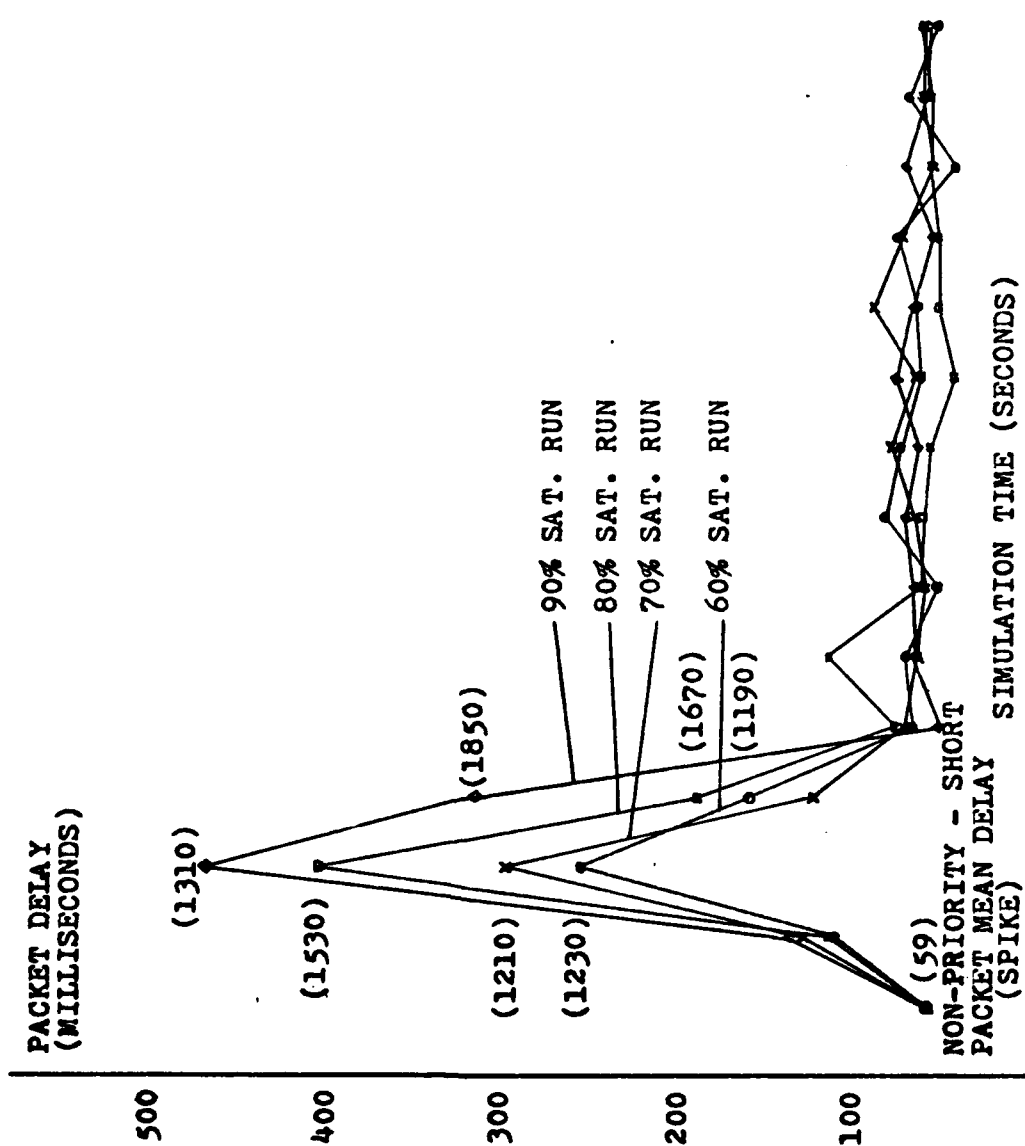


Fig P-2. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)



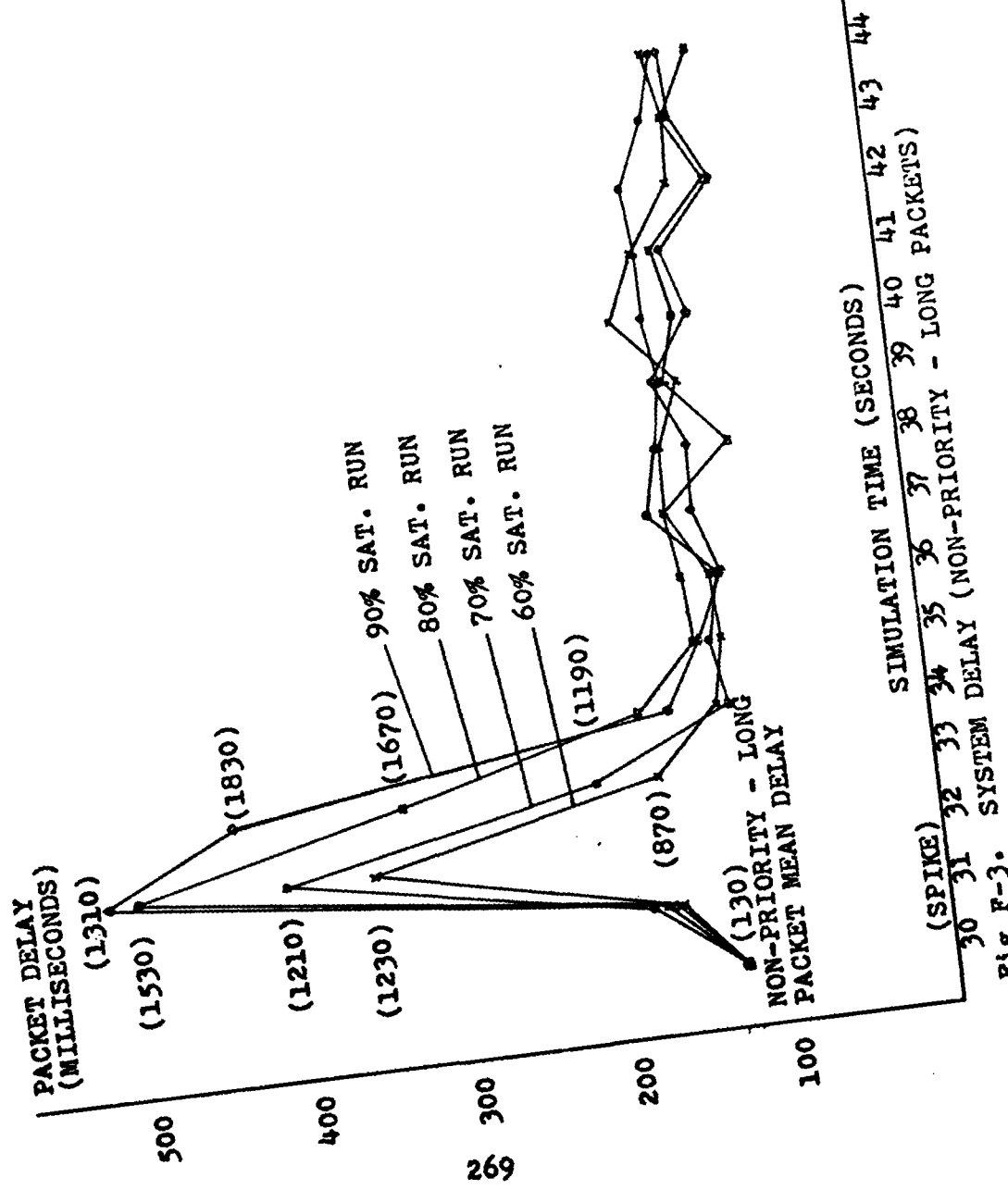


Fig F-3.

## TABLE F-I

Simulation Time (seconds)	Delay Times (milliseconds)							
	60% Sat. Mean	60% Sat. Run Max	70% Sat. Mean	70% Sat. Run Max	80% Sat. Mean	80% Sat. Run Max	90% Sat. Mean	90% Sat. Run Max
30	66	230	66	230	66	230	66	230
31 (spike time)	-	-	45	45	96	110	48	90
32	-	-	-	-	-	-	65	130
33	123	123	-	-	79	79	33	33
34	33	33	-	-	3	3	-	-
35	125	125	-	-	-	-	-	-
36	62	70	89	89	-	-	63	63
37	-	-	38	38	-	-	-	-
38	71	71	77	77	-	-	-	-
39	3	3	-	-	-	-	18	30
40	-	-	-	-	-	-	-	-
41	-	-	-	-	-	-	-	-
42	-	-	-	-	63	63	-	-
43	33	33	-	-	-	-	-	-
44	-	-	-	-	3	3	85	85

System Delay (priority - long packets)

271

Appendix G  
Graphical and Tabular Delay Results  
From the Trunk Loss Runs

Albany - Ft. Detrick Trunk Loss

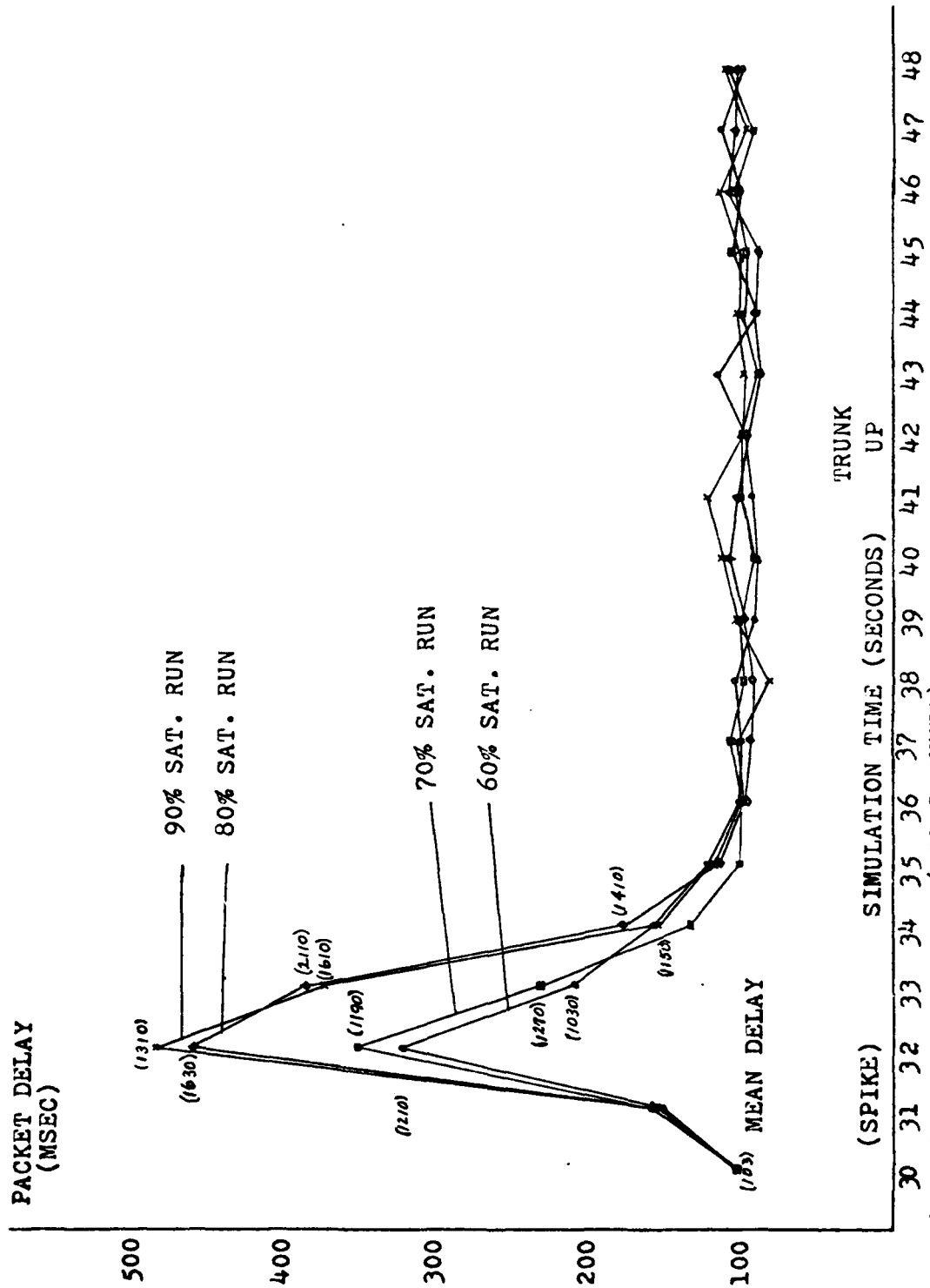


Fig G-1. SYSTEM DELAY (ALL PACKETS)

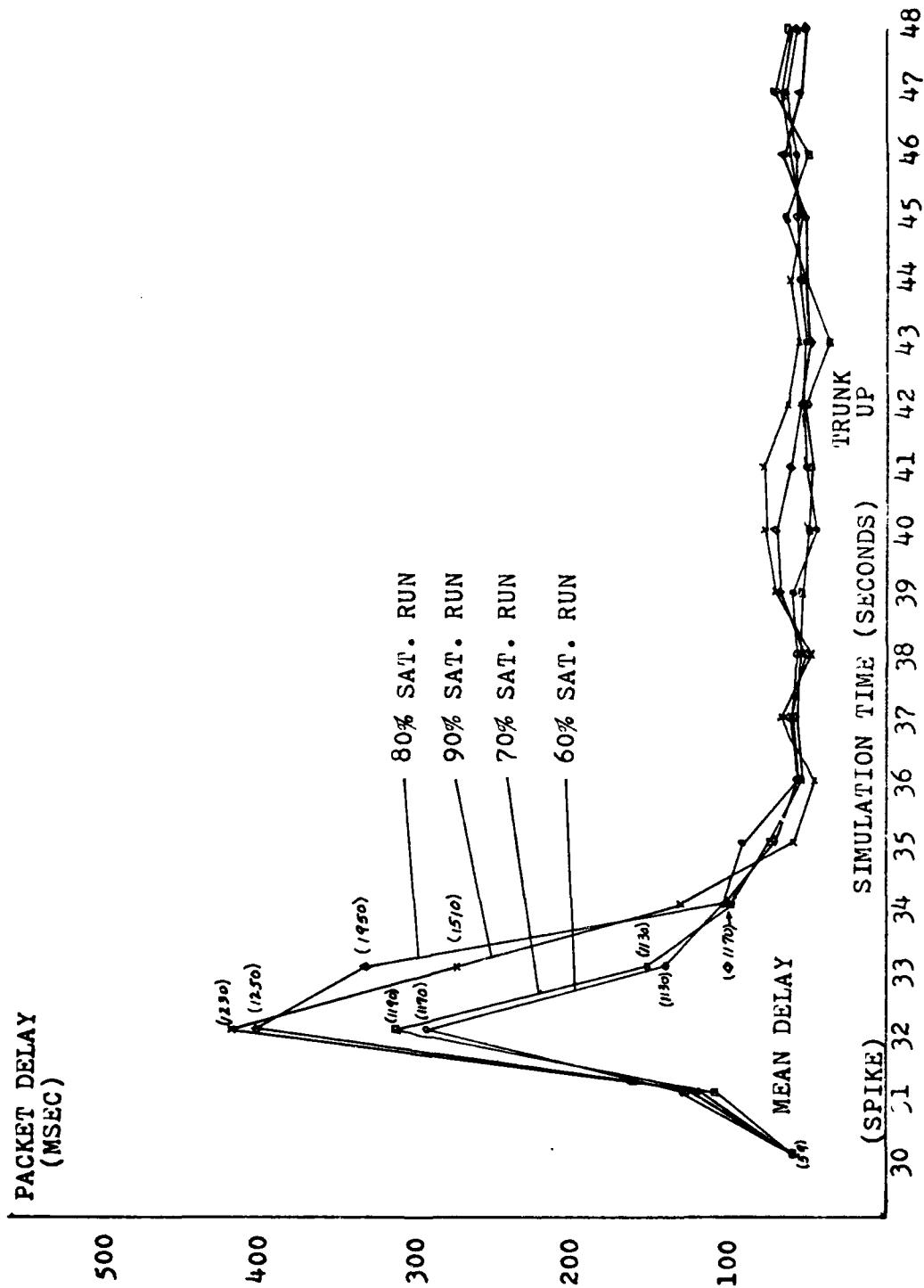


Fig G-2. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

AD-A080 484 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/O 17/2  
AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II.(U)

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/O 17/2  
AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II.(U)

UNCLASSIFIED DEC 79 J A WHITT  
AFIT/OC5/EE/79-15

AFIT/OC5/EE/79-15

NL

4 or 5

Δ<sub>2</sub>  
ΔORC4+Δ



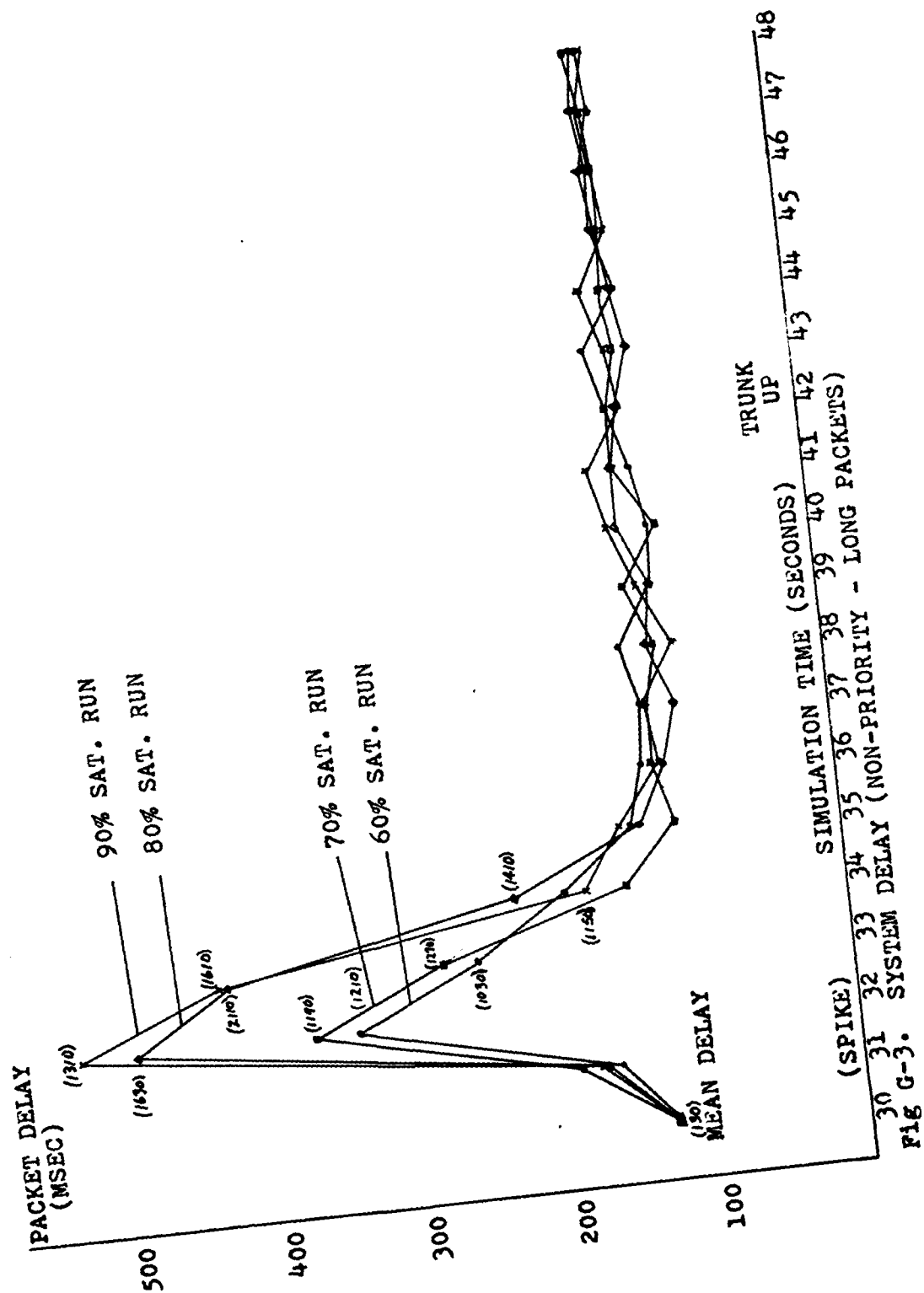


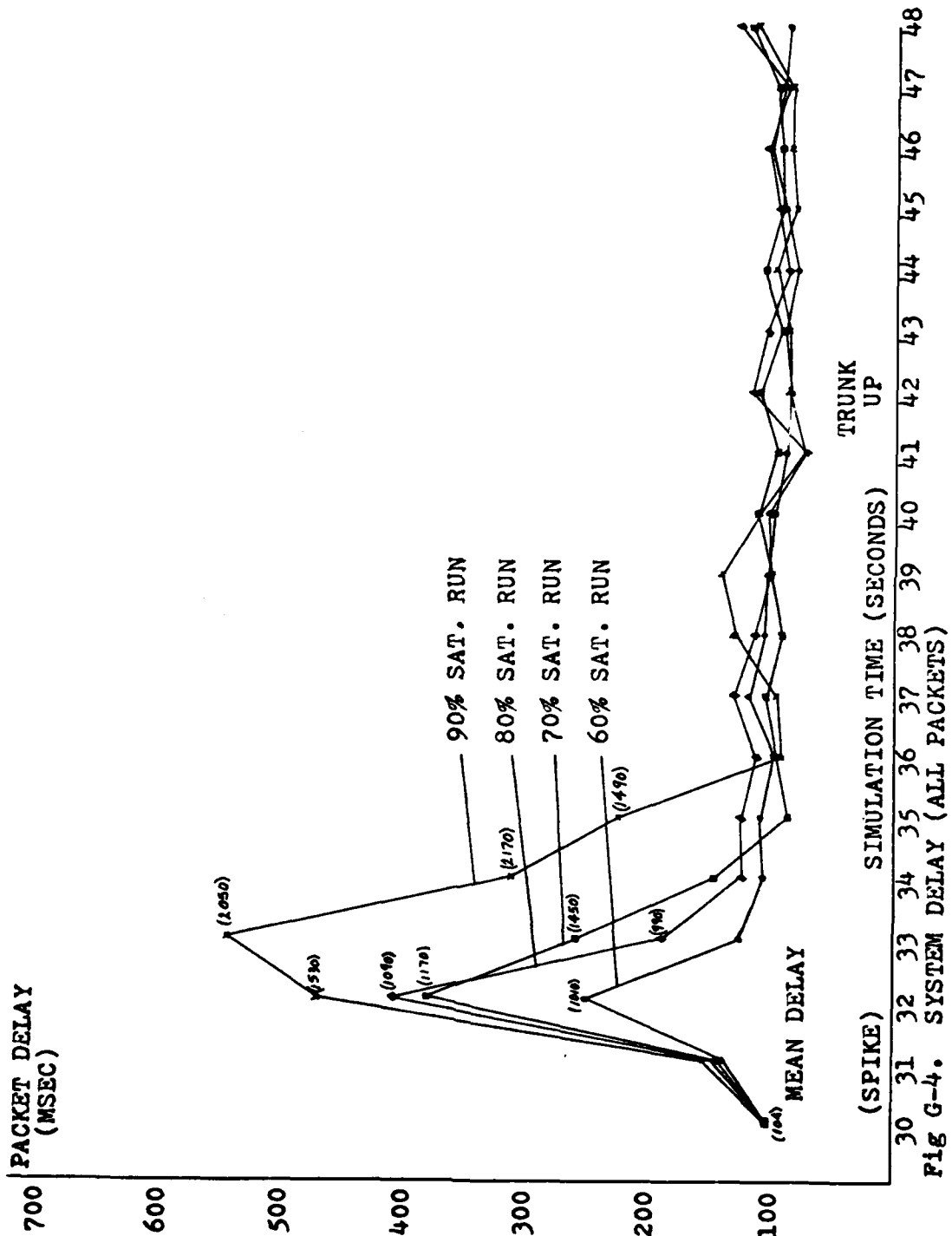
TABLE G-I  
System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)				90% Sat.			
	60% Sat. Mean	70% Sat. Mean	80% Sat. Mean	Run Max	Mean	Run Max	Mean	Run Max
30	66	66	66	230	66	230	66	230
31 (spike time)	-	-	152	-	64	90	-	-
32	-	130	-	130	-	-	-	-
33	-	89	35	110	117	117	117	117
34	33	-	136	-	33	136	33	33
35	-	-	3	-	41	70	41	70
36	18	58	-	70	-	-	-	-
37	-	117	67	130	-	67	-	-
38	-	-	40	-	-	40	-	-
39	-	-	-	-	-	-	-	-
40	42	124	-	124	-	-	-	-
41	-	-	142	-	-	142	-	-
42 (trunk up)	-	-	106	-	-	106	-	-
43	-	35	36	35	3	36	3	3
44	-	-	-	-	33	-	33	33
45	-	-	51	-	-	70	-	47
46	33	33	-	33	47	-	47	180
47	-	57	47	80	160	47	160	35
48	-	-	3	-	35	3	35	-

TABLE G-II  
System Delay (priority - long packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.			
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max
30	122	250	122	250	122	250	122	250	122	250	122	250	122	250	122	250
31 (spike time)	101	250	-	-	-	-	61	90	61	90	61	90	151	250	151	250
32	6	6	159	210	6	210	6	10	6	10	6	10	158	350	158	350
33	3	3	111	111	111	111	266	266	266	266	266	266	351	351	351	351
34	218	230	55	110	3	110	3	3	3	3	3	3	65	130	65	130
35	-	-	154	154	208	154	208	310	208	310	208	310	-	-	-	-
36	-	-	106	106	3	106	3	3	3	3	3	3	-	-	-	-
37	-	-	-	-	126	-	126	250	126	250	126	250	-	-	-	-
38	69	130	183	210	101	210	101	190	101	190	101	190	106	106	106	106
39	178	250	154	310	55	310	55	110	55	110	55	110	-	-	-	-
40	-	-	159	210	209	210	209	209	209	209	209	209	-	-	-	-
41	-	-	164	164	102	164	102	190	102	190	102	190	159	210	159	210
42 (trunk up)	8	8	-	-	36	-	36	70	36	70	36	70	-	-	-	-
43	106	106	-	-	-	-	-	-	-	-	-	-	3	3	3	3
44	106	106	-	-	-	-	-	-	-	-	-	-	106	106	106	106
45	-	-	-	-	215	-	215	290	215	290	215	290	-	-	-	-
46	106	106	106	106	-	106	-	-	-	-	-	-	291	291	291	291
47	66	66	-	-	71	-	71	71	71	71	71	71	150	190	150	190
48	-	-	81	81	213	81	213	213	213	213	213	213	37	37	37	37

Andrews - Tinker Trunk Loss



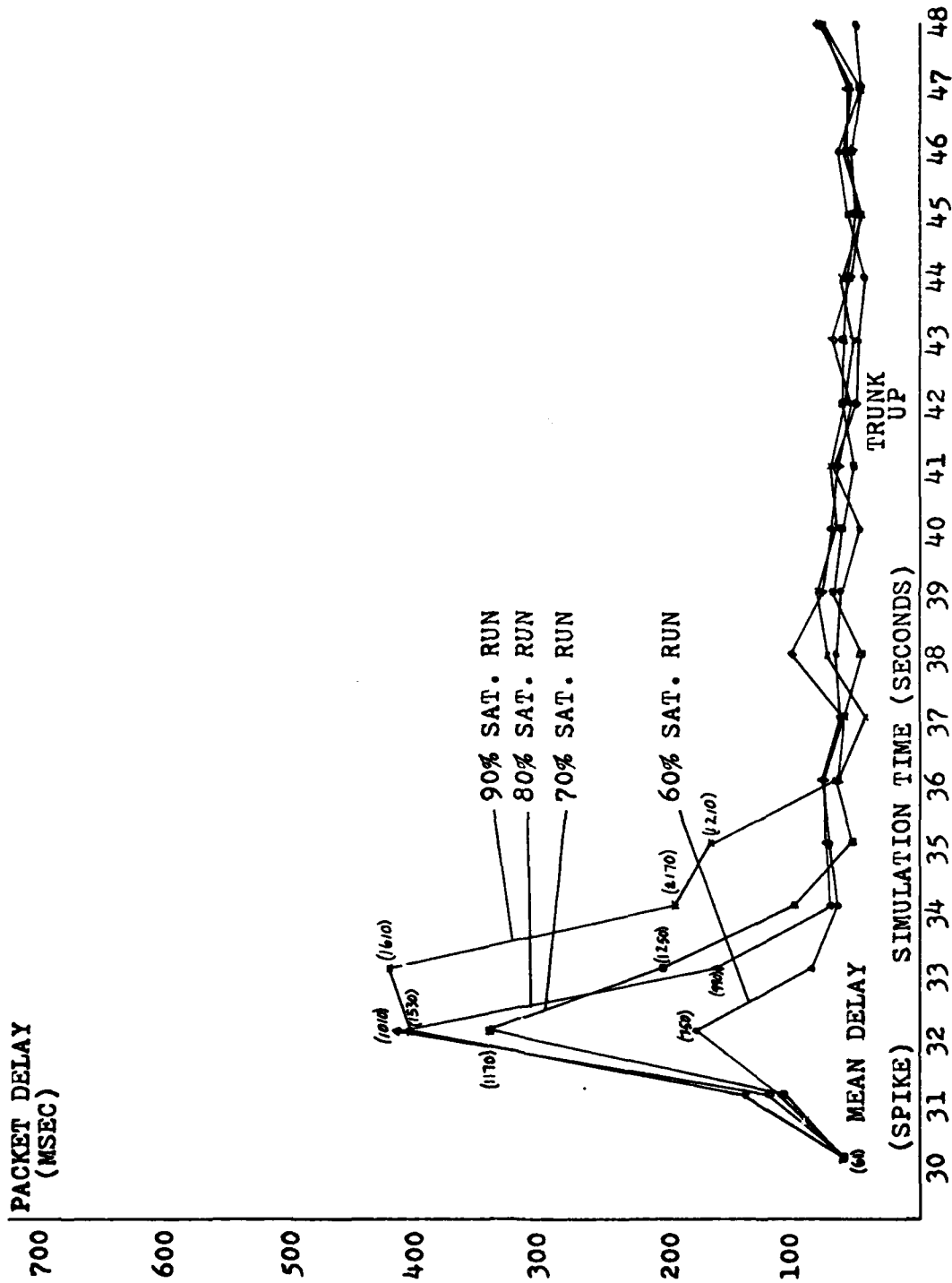


Fig G-5. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

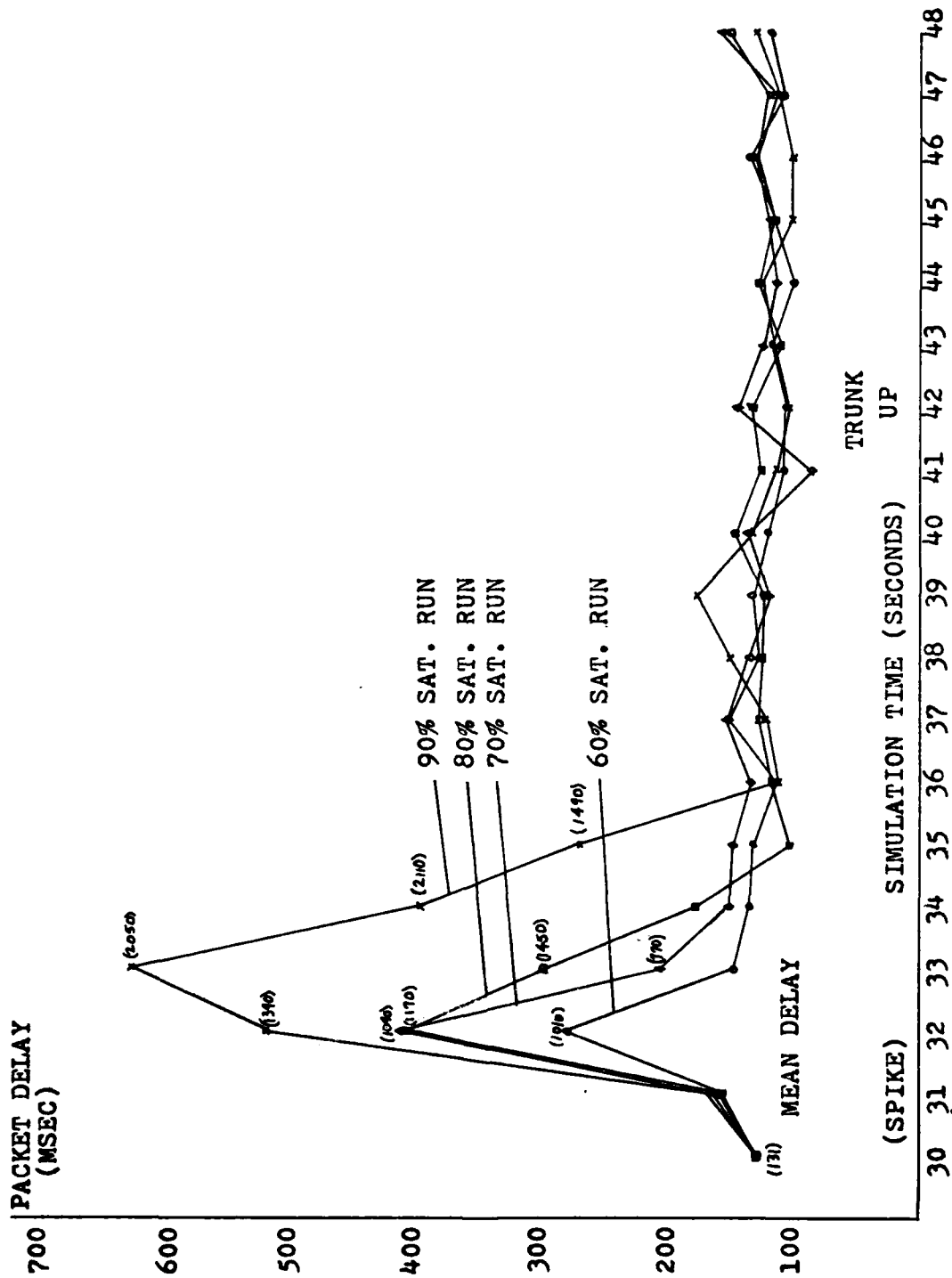


TABLE G-III  
System Delay (priority - short packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	72	230	72	230	72	230	72	230	72	230	72	230	72	230
31 (spike time)	47	90	49	90	66	150	66	150	73	130	73	130	73	130
32	-	-	-	-	48	50	48	50	103	103	103	103	103	103
33	-	-	67	170	-	-	-	-	-	-	-	-	-	-
34	41	50	145	145	-	-	-	-	3	3	3	3	3	3
35	33	33	-	-	40	70	40	70	-	-	-	-	-	-
36	-	-	40	40	108	110	108	110	67	67	67	67	67	67
37	-	-	-	-	63	70	63	70	-	-	-	-	-	-
38	87	87	-	-	39	39	39	39	-	-	-	-	-	-
39	-	-	3	3	77	77	77	77	-	-	-	-	-	-
40	-	-	-	-	-	-	-	-	-	-	-	-	-	-
41	-	-	-	-	-	-	-	-	60	60	60	60	60	60
42 (trunk up)	-	-	-	-	-	-	-	-	-	-	-	-	-	-
43	3	3	-	-	-	-	-	-	-	-	-	-	-	-
44	-	-	37	37	33	33	33	33	35	35	35	35	35	35
45	-	-	73	73	-	-	-	-	-	-	-	-	-	-
46	18	30	3	3	-	-	-	-	-	-	-	-	-	-
47	35	35	-	-	131	131	131	131	-	-	-	-	-	-
48	103	103	3	3	-	-	-	-	-	-	-	-	-	-



TABLE G-IV  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)						Delay Times (milliseconds)					
	60% Sat.		70% Sat.		80% Sat.		60% Sat.		70% Sat.		80% Sat.	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
30	122	250	122	250	122	250	122	250	122	250	122	250
31 (spike time)	168	250	138	290	97	210	97	210	95	210	95	210
32	359	359	-	-	205	330	205	330	115	115	115	115
33	-	-	61	170	72	130	72	130	37	37	37	37
34	-	-	3	3	197	230	197	230	-	-	-	-
35	108	210	131	131	-	-	-	-	259	259	259	259
36	55	110	3	3	-	-	-	-	72	150	72	150
37	-	-	-	-	209	209	209	209	-	-	-	-
38	185	210	357	357	3	3	3	3	3	3	3	3
39	-	-	73	110	-	-	-	-	-	-	-	-
40	61	110	298	298	117	130	117	130	-	-	-	-
41	-	-	149	149	106	106	106	106	-	-	-	-
42 (trunk up)	106	106	109	109	-	-	-	-	-	-	-	-
43	146	146	106	106	3	3	3	3	-	-	-	-
44	222	222	-	-	-	-	-	-	109	109	109	109
45	-	-	73	210	-	-	-	-	106	106	106	106
46	5	5	110	110	-	-	-	-	-	-	-	-
47	3	3	182	182	-	-	-	-	-	-	-	-
48	-	-	-	-	-	-	-	-	-	-	-	-

Ft. Detrick - Tinker Trunk Loss

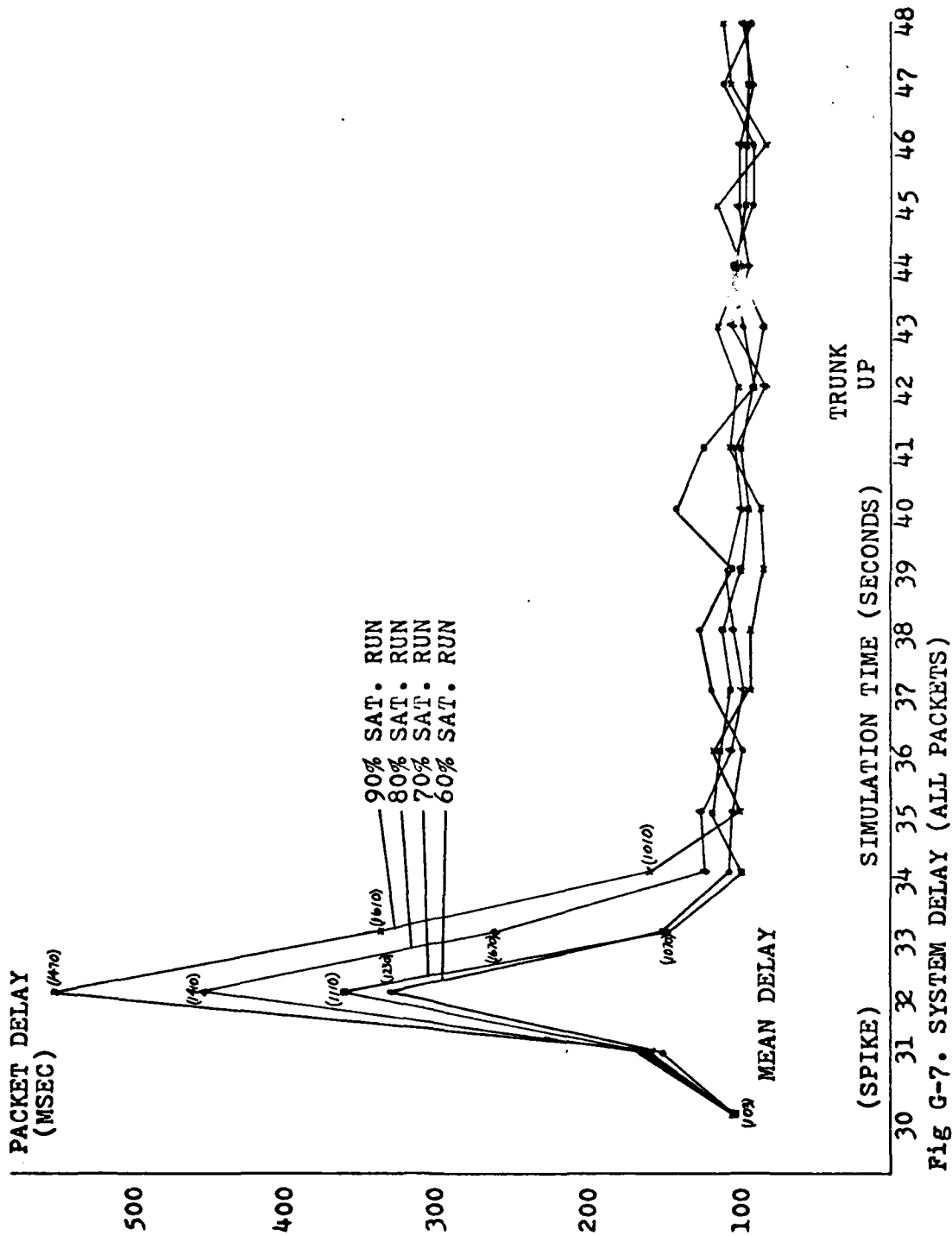
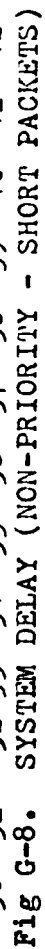


Fig G-7. SYSTEM DELAY (ALL PACKETS)



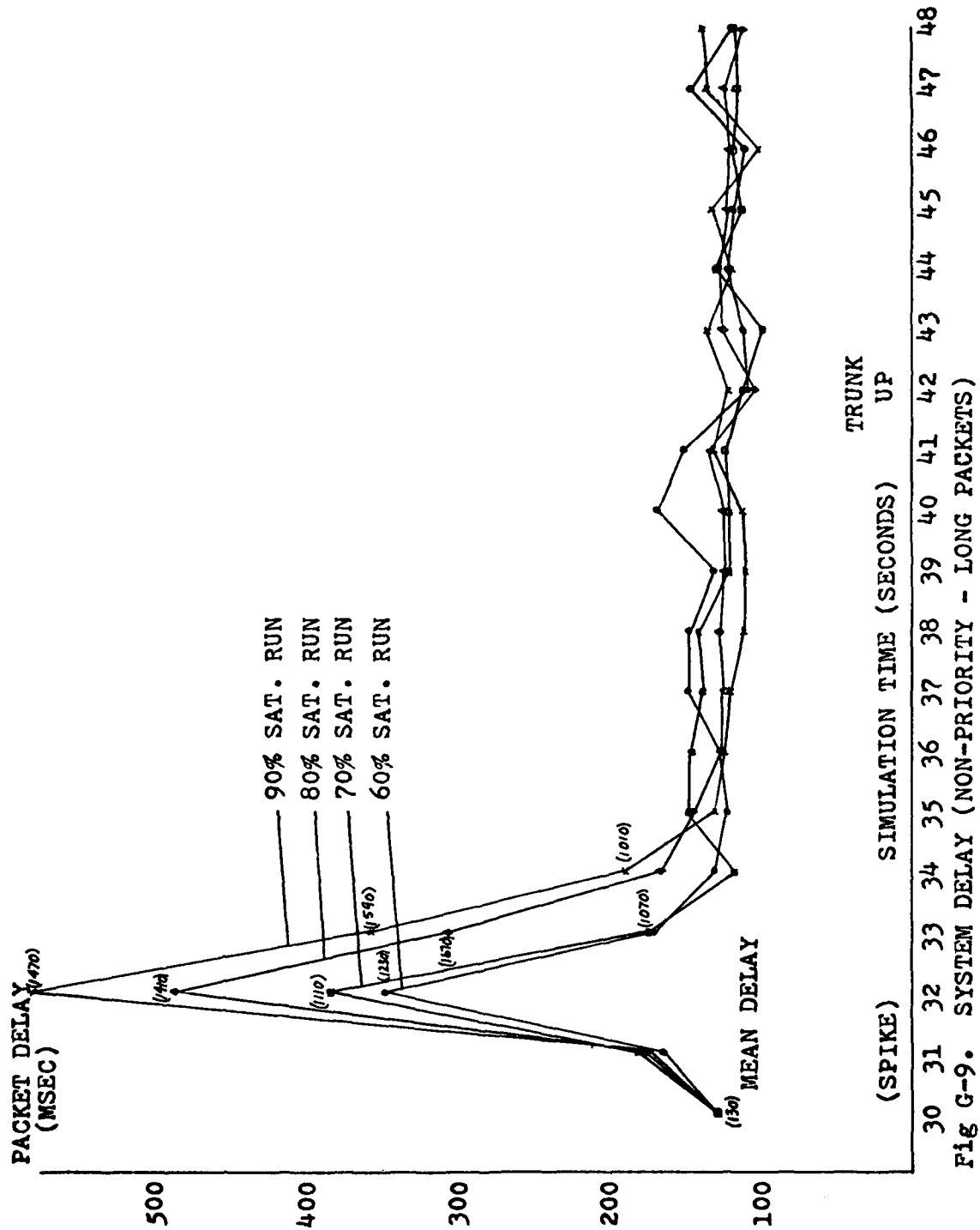


Fig G-9. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE G-V  
System Delay (priority - short packets)

Simulation Times (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	66	230	66	230	66	230	66	230	66	230	66	230	66	230
31 (spike time)	-	-	61	70	-	-	-	-	-	-	56	110	-	-
32	-	-	-	-	-	-	36	36	-	-	-	-	-	-
33	123	123	-	-	33	33	-	-	-	-	-	-	-	-
34	33	33	-	-	-	-	-	-	-	-	-	-	-	-
35	125	125	3	3	-	-	-	-	-	-	27	50	-	-
36	62	70	41	41	107	107	63	63	-	-	79	90	-	-
37	-	-	-	-	-	-	-	-	-	-	-	-	-	-
38	71	71	-	-	-	-	-	-	-	-	3	3	-	-
39	3	3	-	-	-	-	-	-	-	-	-	-	-	-
40	-	-	-	-	-	-	-	-	-	-	-	-	-	-
41	-	-	-	-	-	-	-	-	-	-	-	-	-	-
42 (trunk up)	98	98	-	-	-	-	-	-	-	-	-	-	-	-
43	-	-	-	-	3	3	-	-	-	-	-	-	-	-
44	-	-	-	-	-	-	106	210	-	-	-	-	-	-
45	60	110	34	34	130	130	75	90	-	-	-	-	-	-
46	137	137	-	-	33	33	-	-	-	-	-	-	-	-
47	33	33	-	-	45	50	-	-	-	-	-	-	-	-
48	-	-	3	3	-	-	-	-	-	-	-	-	-	-

TABLE G-VI  
System Delay (priority - long packets)

Simulation Times (seconds)	Delay Times (milliseconds)						90% Sat.	
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	80% Sat. Mean	Run Max	Mean	Run Max
30	122	250	122	250	122	250	122	250
31 (spike time)	101	250	207	250	177	310	163	250
32	3	3	-	-	-	-	-	-
33	108	210	3	3	-	-	3	3
34	-	-	257	257	5	5	-	-
35	159	210	194	270	-	-	108	108
36	55	110	-	-	169	169	143	210
37	205	205	-	-	137	190	-	-
38	-	-	72	150	-	-	164	230
39	57	101	179	230	94	190	152	152
40	150	210	116	116	106	106	-	-
41	-	-	-	-	106	106	190	190
42 (trunk up)	183	183	67	130	3	3	228	228
43	-	-	3	3	4	4	-	-
44	233	233	167	210	120	120	85	230
45	3	3	152	310	-	-	209	209
46	163	163	106	106	106	106	3	3
47	-	-	59	90	3	3	50	70
48	-	-	-	-	209	209	106	106

Gentile - Ft. Detrick Trunk Loss



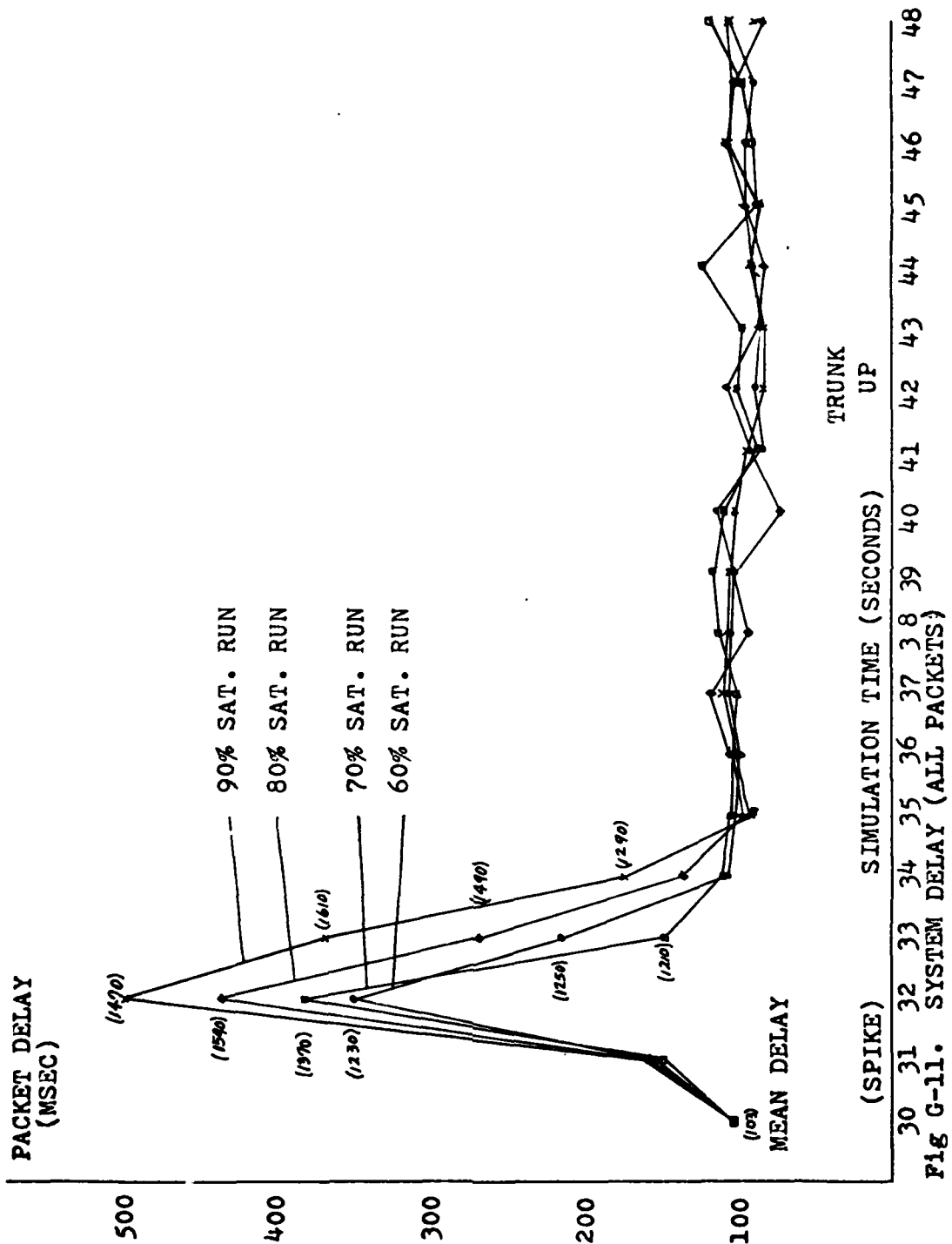


Fig G-11. SYSTEM DELAY (ALL PACKETS)

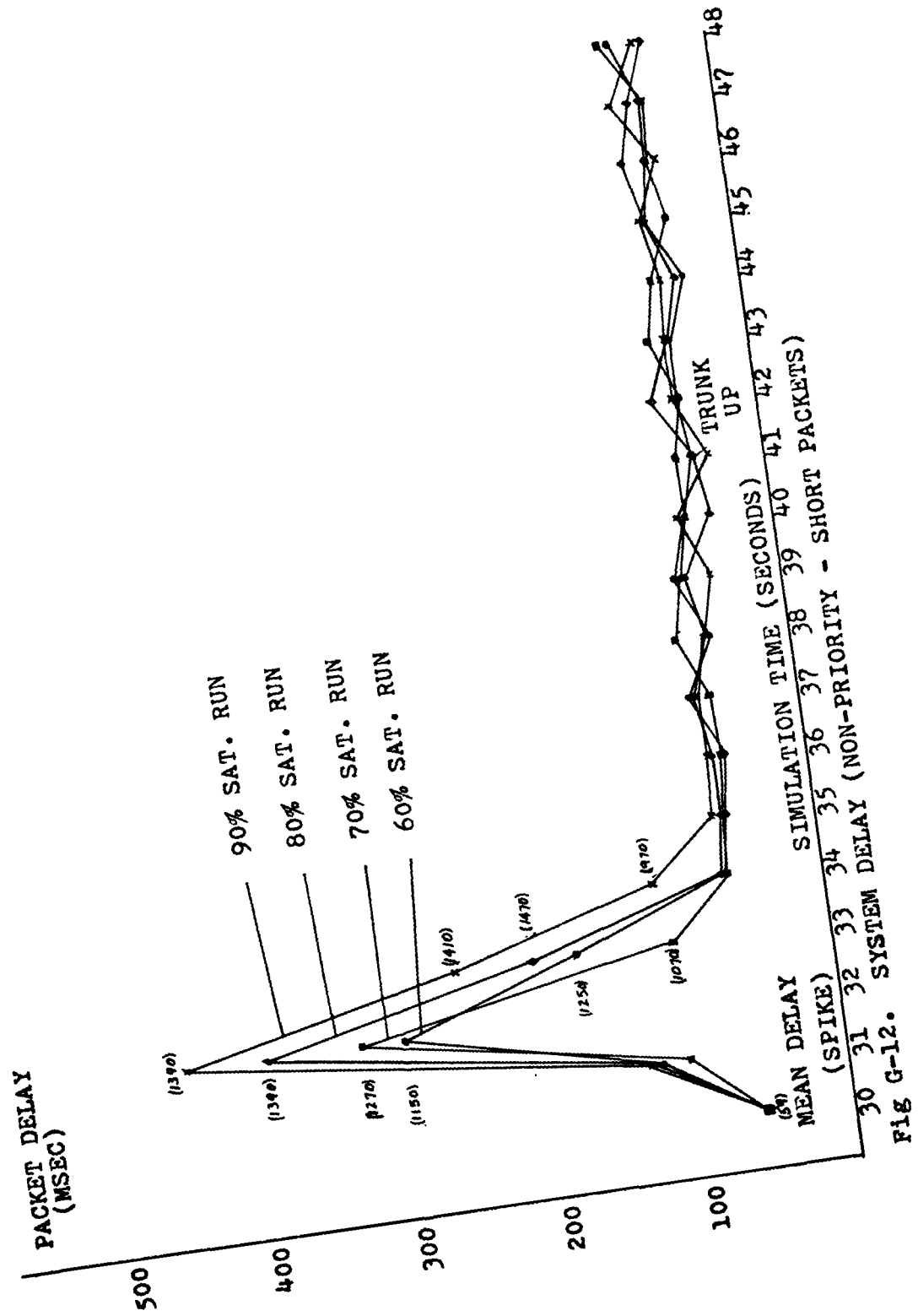


FIG G-12.

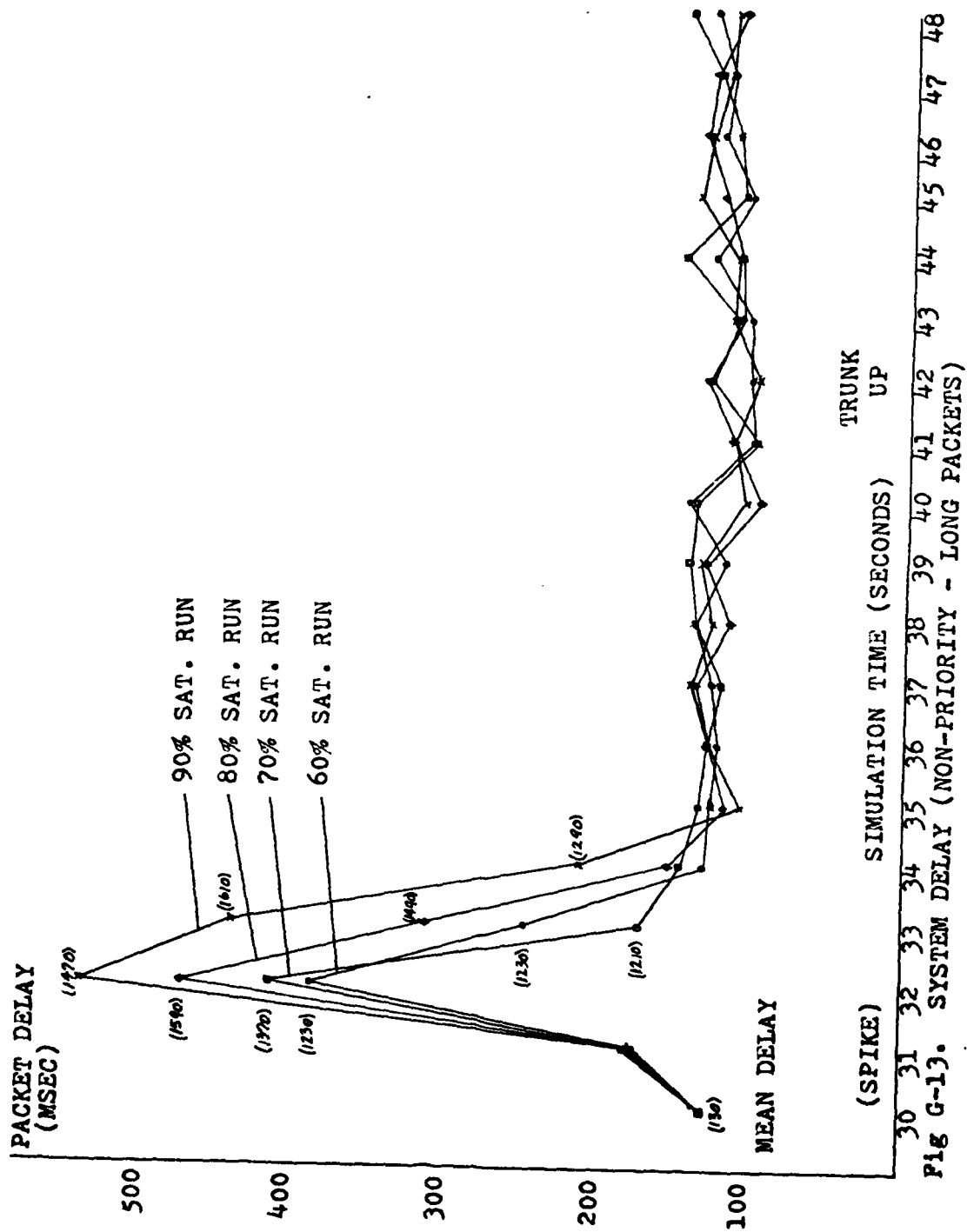


TABLE G-VII  
System Delay (priority - short packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.		Run	
	Mean	Max	Run	Max	Mean	Max	Run	Max	Mean	Max	Run	Max	Mean	Max	Mean	Max
30	66	230	-	-	66	230	-	-	66	230	-	-	66	230	66	230
31 (spike time)	-	-	-	-	-	-	-	-	-	-	-	-	137	250	137	250
32	44	90	90	90	162	162	162	162	33	33	33	33	128	128	128	128
33	33	33	33	33	33	33	33	33	3	3	3	3	152	190	152	190
34	-	-	-	-	3	3	3	3	41	41	41	41	173	173	173	173
35	-	-	-	-	-	-	-	-	8	8	8	8	94	130	94	130
36	-	-	-	-	28	28	50	50	-	-	-	-	151	151	151	151
37	67	130	130	130	-	-	-	-	68	68	68	68	-	-	-	-
38	-	-	-	-	35	35	35	35	-	-	-	-	108	110	108	110
39	-	-	-	-	33	33	33	33	-	-	-	-	-	-	-	-
40	5	5	5	5	3	3	3	3	38	38	38	38	-	-	-	-
41	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
42 (trunk up)	-	-	-	-	33	33	33	33	-	-	-	-	-	-	-	-
43	18	30	30	30	-	-	-	-	-	-	-	-	63	70	63	70
44	-	-	-	-	-	-	-	-	-	-	-	-	67	67	67	67
45	44	44	44	44	63	63	63	63	-	-	-	-	-	-	-	-
46	7	7	7	7	-	-	-	-	47	47	47	47	-	-	-	-
47	63	63	63	63	33	33	33	33	18	18	18	18	91	170	91	170
48	-	-	-	-	100	110	110	110	-	-	-	-	-	-	-	-

TABLE VIII  
System Delay (priority - long packets)

Simulation Times (seconds)	Delay Times (milliseconds)				80% Sat.				90% Sat.	
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	122	250	122	250	122	250	122	250	122	250
31 (spike time)	126	250	176	250	258	310	48	90	48	90
32	-	-	212	212	83	130	123	123	123	123
33	136	330	213	250	121	121	65	65	65	65
34	171	250	-	-	80	130	19	30	19	30
35	126	150	57	290	-	-	-	-	-	-
36	-	-	-	-	106	106	187	190	187	190
37	134	170	172	230	178	290	-	-	-	-
38	-	-	64	130	106	106	-	-	-	-
39	143	150	217	270	67	330	80	150	80	150
40	106	106	-	-	0	-	106	106	106	106
41	308	308	252	252	252	252	106	106	106	106
42 (trunk up)	-	-	149	250	110	110	-	-	-	-
43	-	-	-	-	3	3	292	292	292	292
44	3	3	-	-	-	-	133	190	133	190
45	-	-	145	210	176	176	-	-	-	-
46	209	210	-	-	-	-	-	-	-	-
47	-	-	-	-	3	3	106	106	106	106
48	-	-	-	-	109	109	119	130	119	130

Gentile - Tinker Trunk Loss

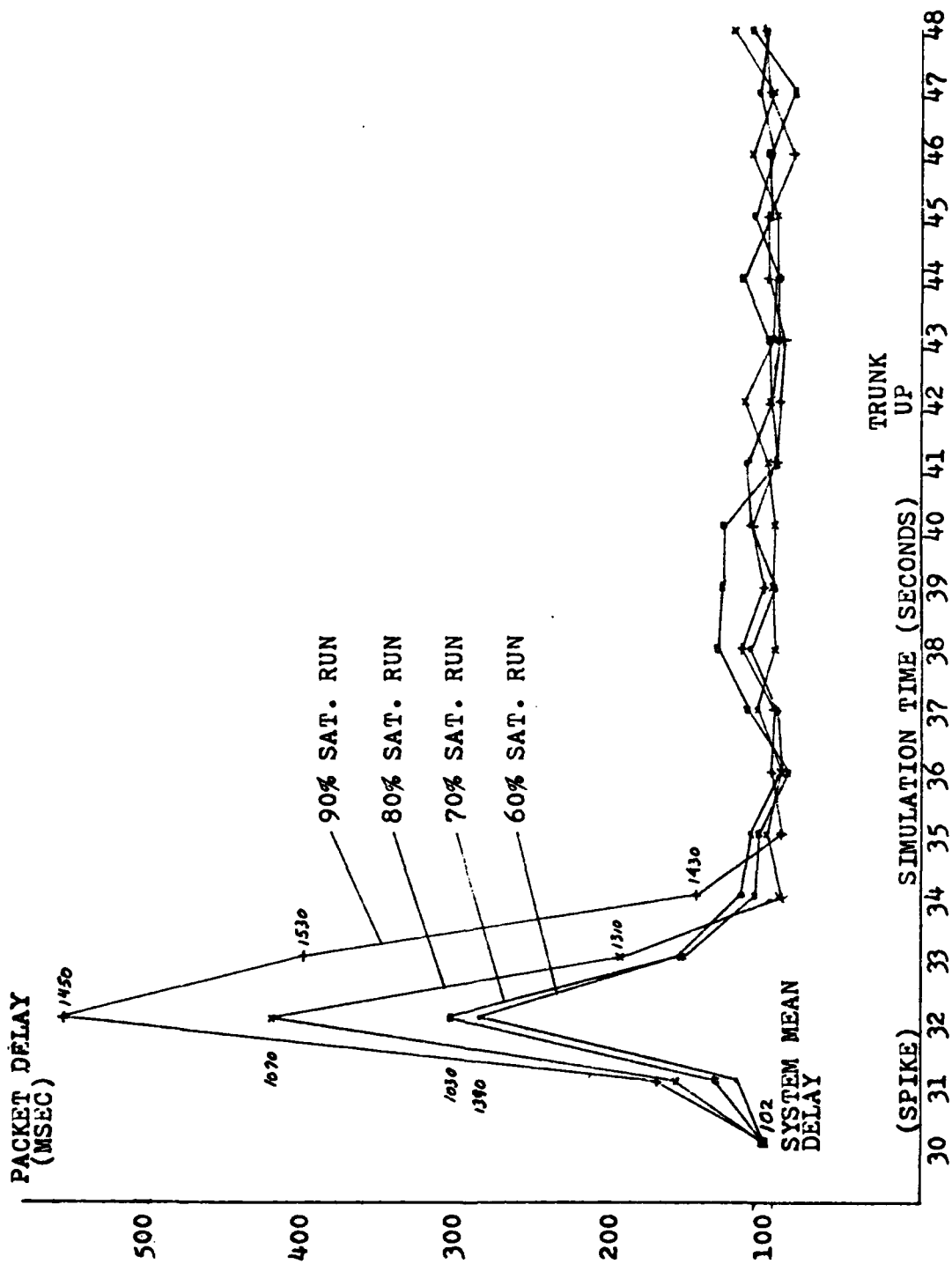


Fig G-14. SYSTEM DELAY (ALL PACKETS)

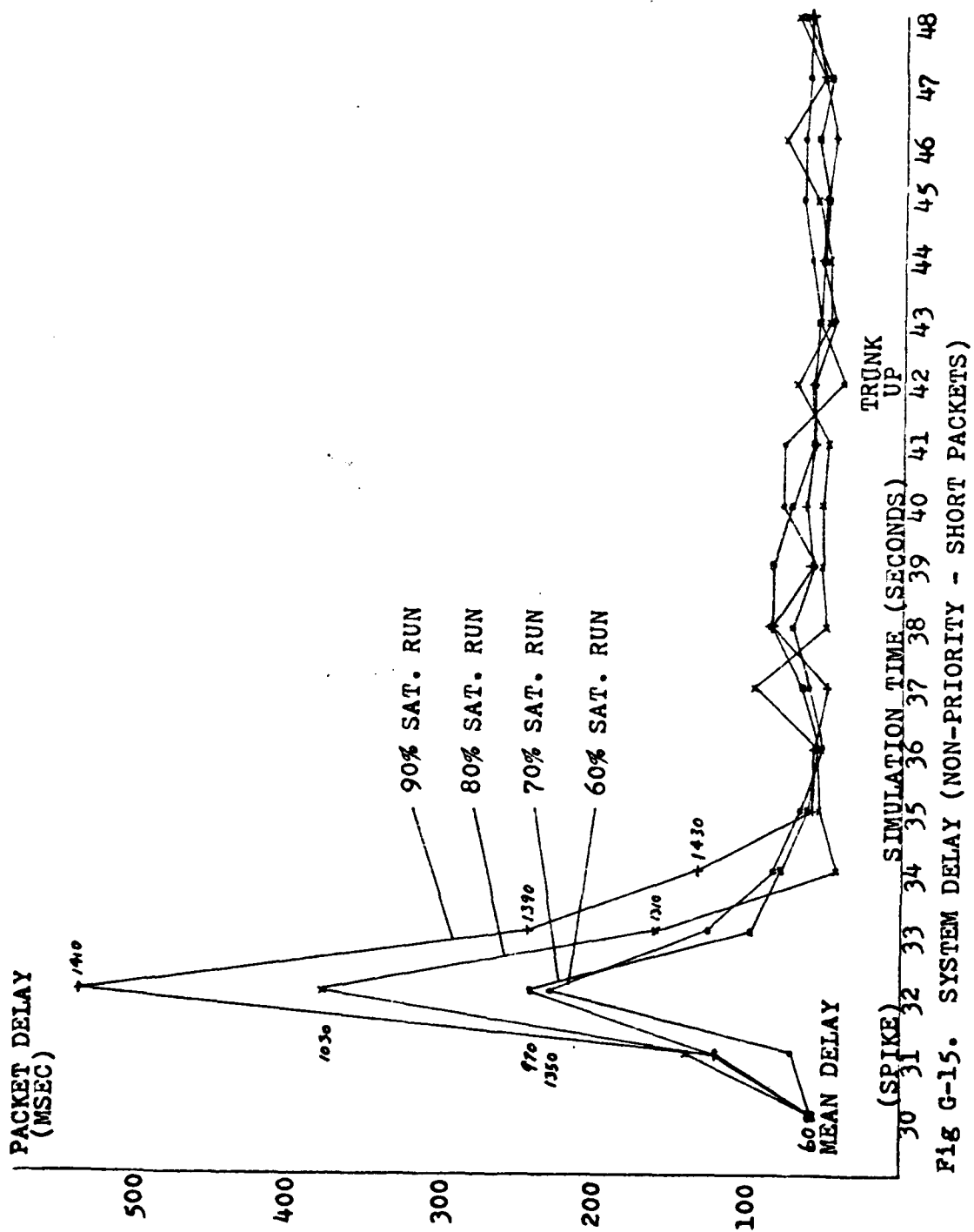


FIG G-15. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)



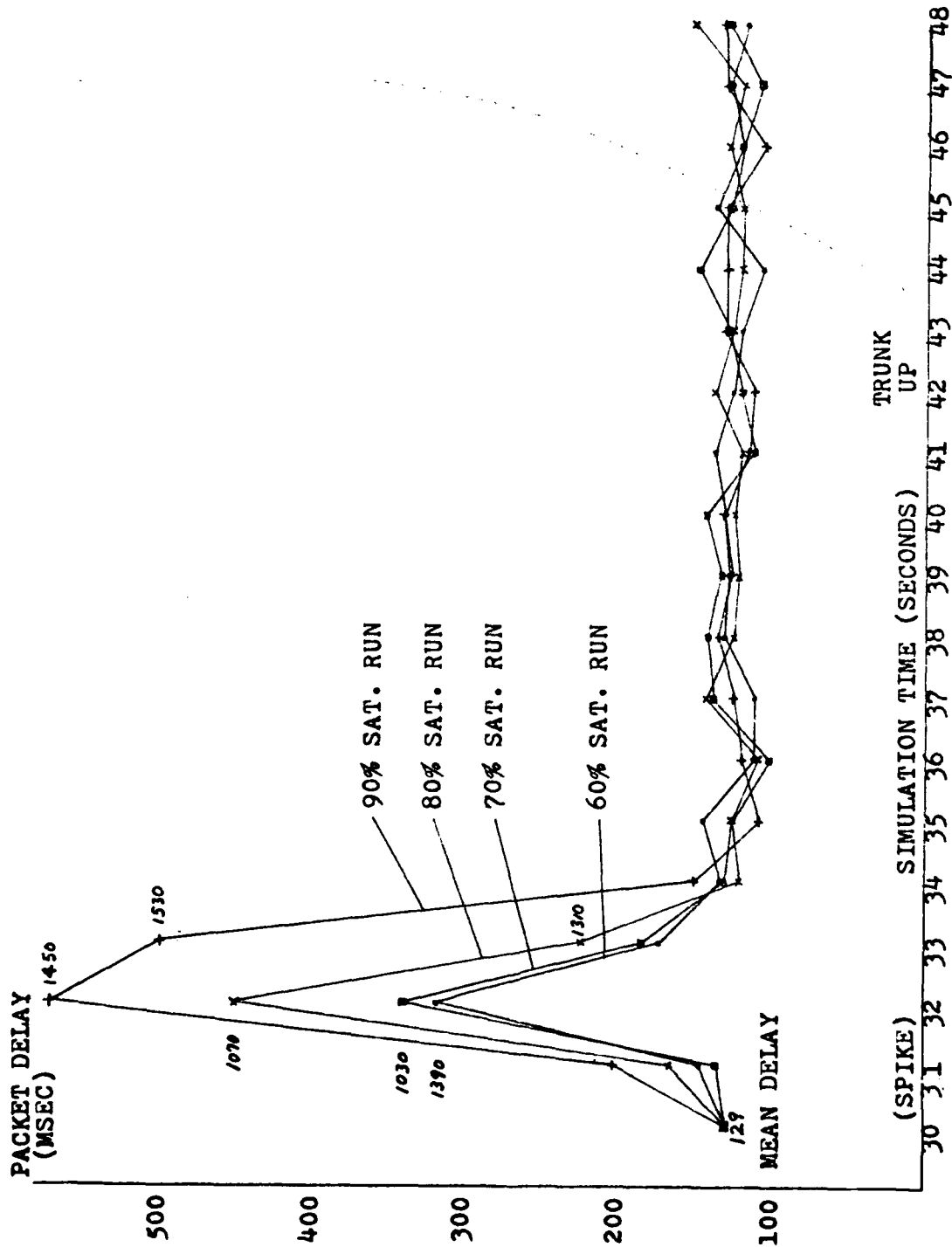


Fig G-16. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE G-IX  
System Delay (priority - short packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.			
	Mean	Max	Run	Max	Mean	Max	Run	Max	Mean	Max	Run	Max	Mean	Max	Run	Max
30	69	230	69	230	69	230	69	230	69	230	69	230	69	230	69	230
31 (spike time)	-	-	46	90	46	90	46	90	84	130	18	30	18	30	18	30
32	-	-	-	-	-	-	-	-	-	-	-	-	115	190	-	-
33	-	-	-	-	-	-	-	-	5	5	-	-	-	-	-	-
34	-	-	-	-	-	-	-	-	18	30	-	-	-	-	-	-
35	-	-	-	-	-	-	-	-	-	-	-	-	33	33	-	-
36	-	-	3	3	3	3	3	3	33	33	-	-	-	-	-	-
37	-	-	-	-	-	-	-	-	3	3	-	-	-	-	-	-
38	-	-	33	33	33	33	33	33	156	156	-	-	33	33	-	-
39	49	49	-	-	-	-	-	-	3	3	-	-	33	33	-	-
40	48	70	103	103	103	103	103	103	-	-	-	-	-	-	-	-
41	3	3	-	-	-	-	-	-	-	-	-	-	3	3	-	-
42 (trunk up)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
43	33	33	-	-	-	-	-	-	-	-	-	-	73	73	-	-
44	-	-	-	-	-	-	-	-	33	33	-	-	-	-	-	-
45	3	3	-	-	-	-	-	-	63	63	-	-	3	3	-	-
46	33	33	-	-	-	-	-	-	33	33	-	-	-	-	-	-
47	3	3	13	30	13	30	13	30	-	-	-	-	33	33	-	-
48	33	33	-	-	-	-	-	-	37	37	-	-	3	3	-	-

TABLE G-X  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)									
	60% Sat.		70% Sat.		80% Sat.		90% Sat.		Run	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
30	110	250	110	250	110	250	110	250	110	250
31 (spike time)	103	190	193	290	135	250	156	370	156	370
32	-	-	-	-	109	110	111	210	111	210
33	139	310	109	109	120	270	5	5	5	5
34	171	210	-	-	108	210	-	-	-	-
35	114	114	277	277	106	106	106	106	106	106
36	157	157	-	-	271	271	-	-	-	-
37	3	3	55	110	134	170	158	210	158	210
38	-	-	-	-	106	106	203	203	203	203
39	3	3	129	270	56	110	-	-	-	-
40	106	106	156	210	138	210	-	-	-	-
41	3	3	-	-	-	-	-	-	-	-
42 (trunk up)	106	106	-	-	-	-	-	-	-	-
43	106	106	209	209	-	-	132	170	132	170
44	114	114	106	106	128	150	106	106	106	106
45	55	110	153	210	153	210	3	3	3	3
46	106	106	117	130	-	-	-	-	-	-
47	106	106	188	188	225	225	-	-	-	-
48	222	222	-	-	-	-	193	193	193	193

Appendix H  
Graphical and Tabular Delay Results  
From the Link Loss Runs

Albany - Ft. Detrick Link Loss

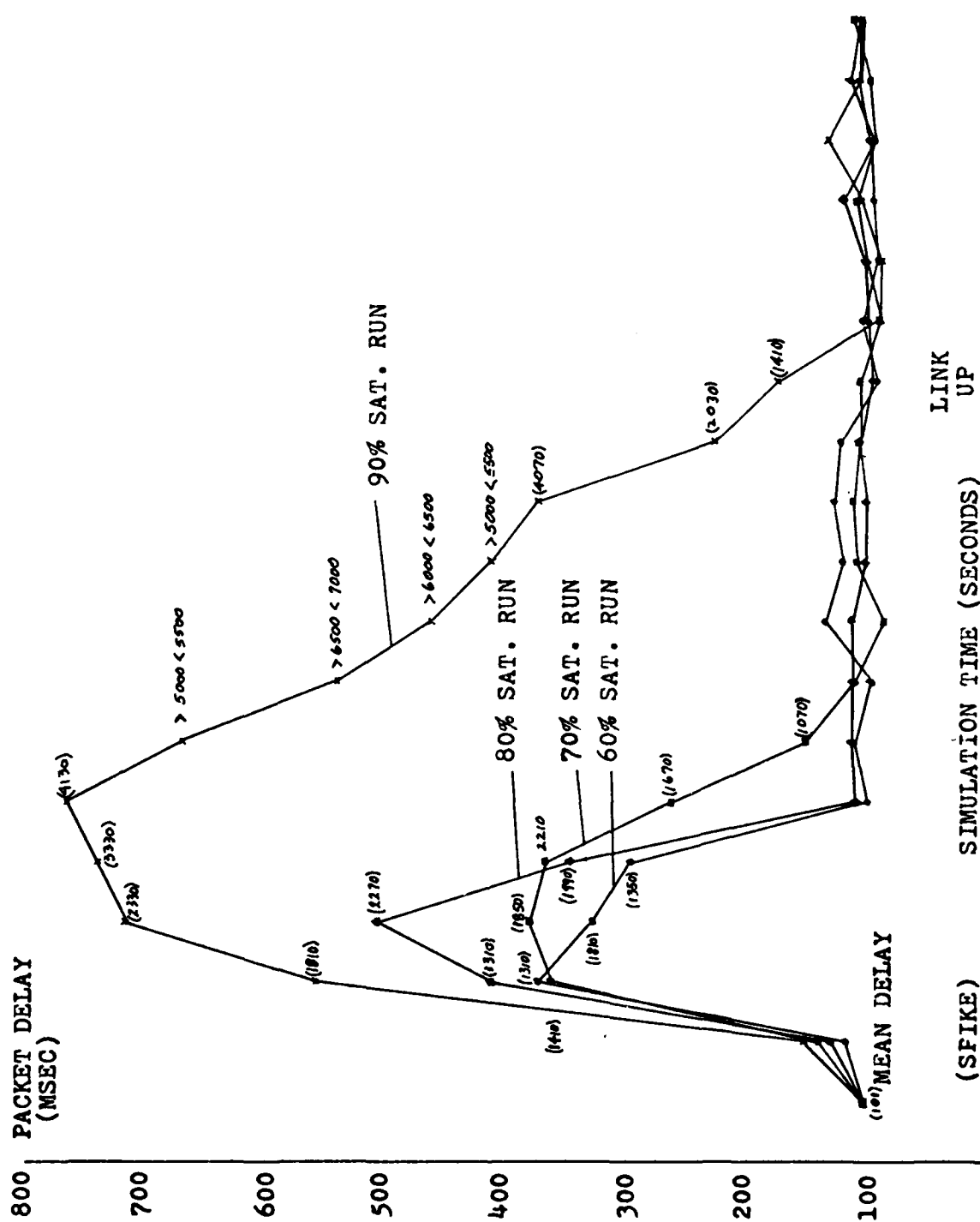


Fig H-1. SYSTEM DELAY (ALL PACKETS)

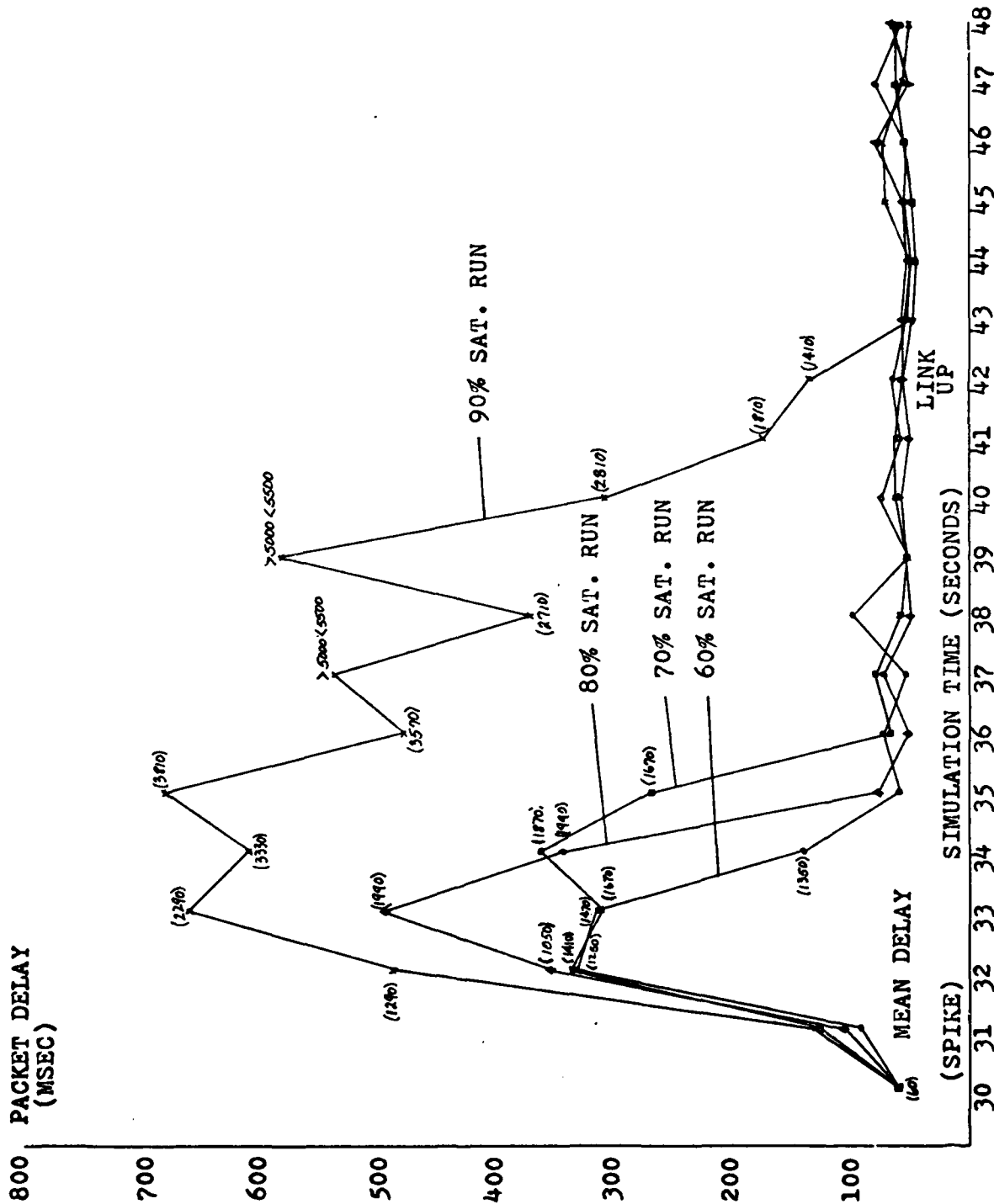


Fig H-2. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

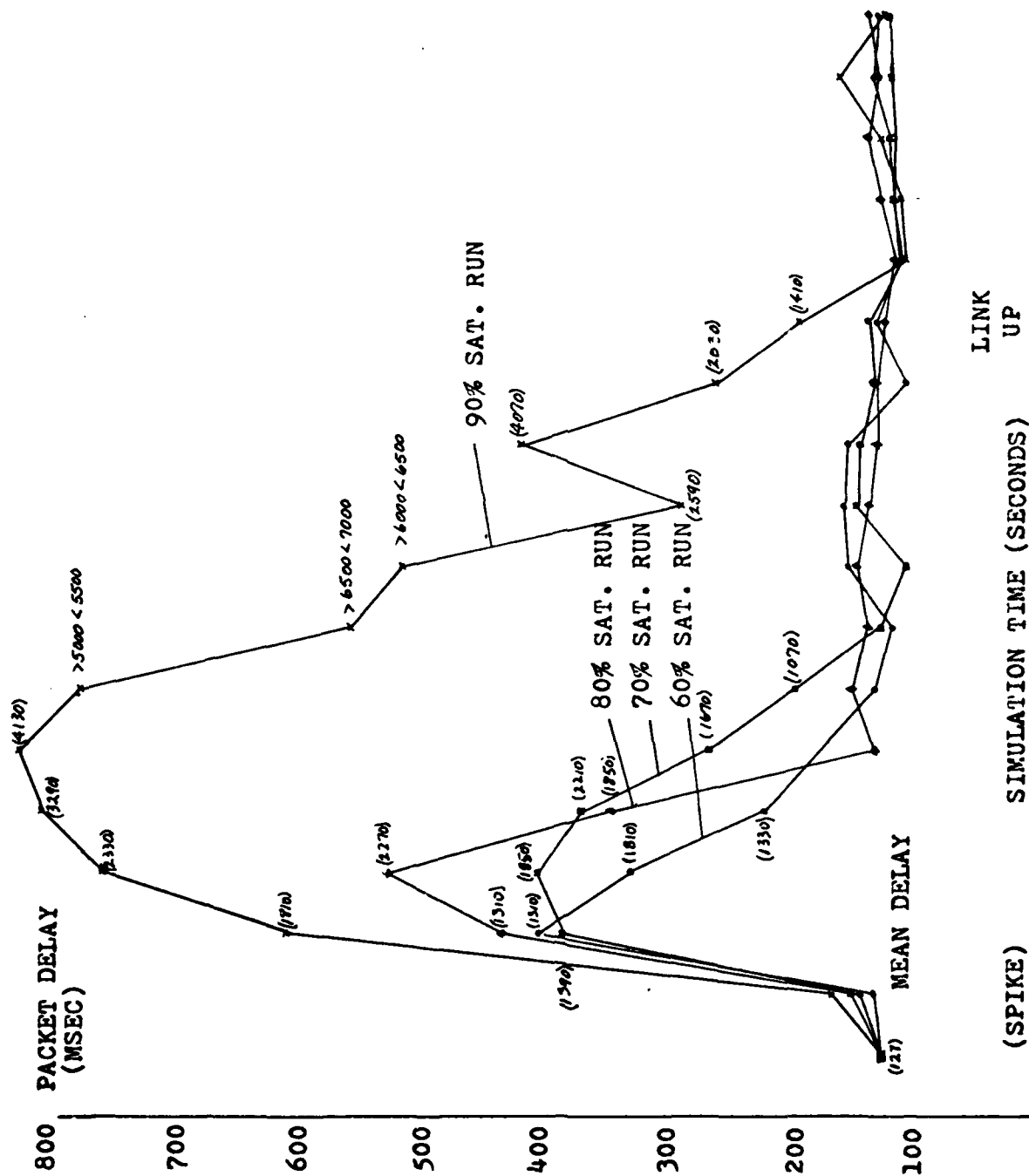


Fig H-3. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)



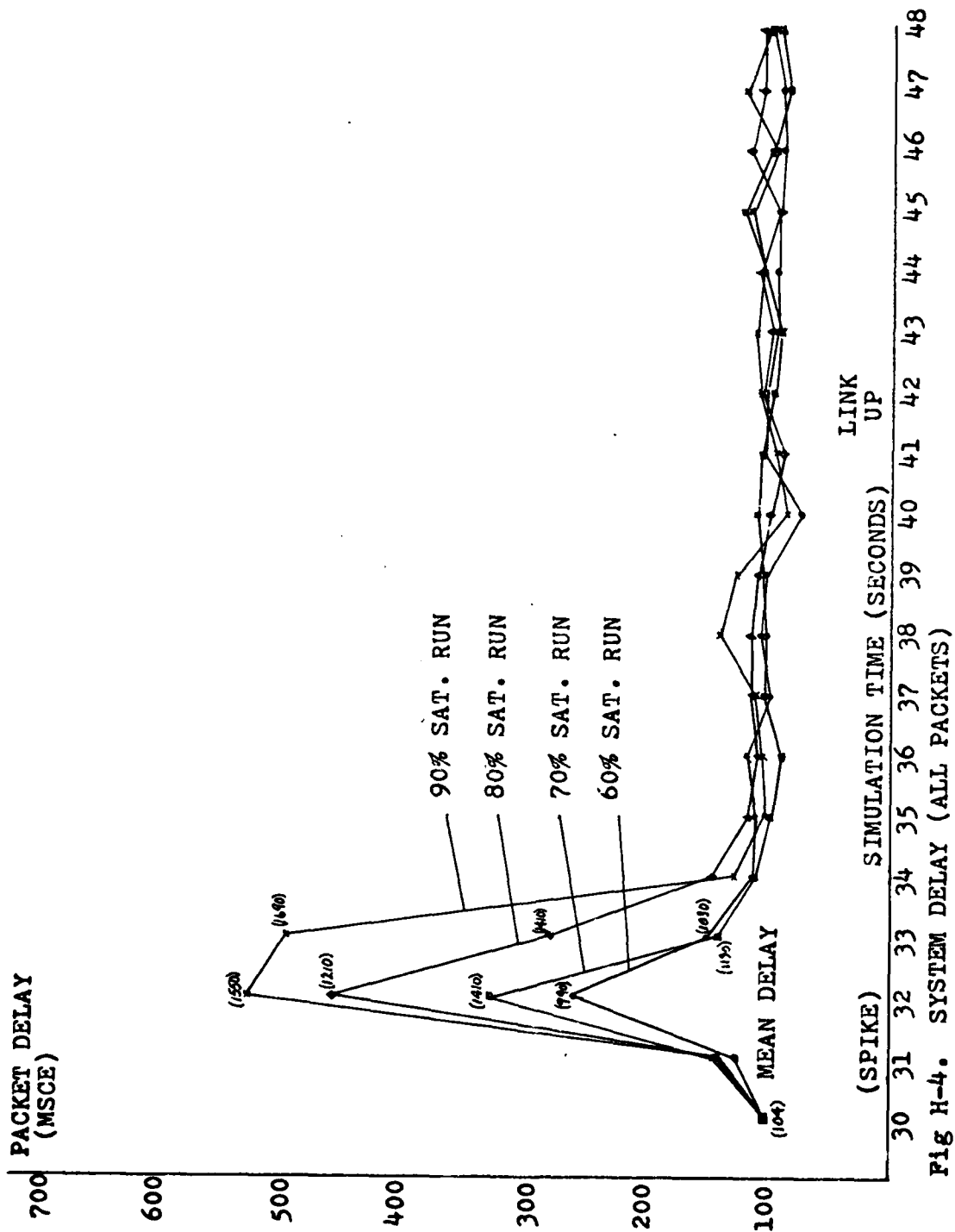
TABLE H-I  
System Delay (priority - short packets)

Simulation Times (seconds)	Delay Times (milliseconds)				90% Sat.			
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	Mean	Run Max	Mean	Run Max
30	65	230	65	230	65	230	65	230
31 (spike time)	-	-	84	90	89	190	103	130
32	-	-	125	150	-	-	-	-
33	-	-	58	58	8	10	-	-
34	-	-	83	83	-	-	83	150
35	43	43	37	50	60	110	48	48
36	33	33	109	150	49	70	-	-
37	64	64	-	-	-	-	66	190
38	33	33	-	-	-	-	-	-
39	-	-	-	-	3	3	35	35
40	66	66	-	-	-	-	117	117
41	101	110	-	-	-	-	-	-
42 (link up)	-	-	3	3	18	30	60	60
43	-	-	3	3	-	-	75	110
44	-	-	-	-	-	-	35	35
45	-	-	-	-	-	-	3	3
46	27	50	-	-	-	-	-	-
47	-	-	-	-	-	-	-	-
48	87	87	6	6	87	87	87	87

TABLE H-II  
System Delay (priority - long packets)

Simulation Times (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max
30	117	370	117	370	117	370	117	370	117	370	117	370	117	370
31 (spike time)	250	250	77	61	124	130	130	130	130	130	130	130	153	310
32	226	250	3	3	-	-	-	-	-	-	-	-	149	270
33	3	3	-	-	106	106	106	106	106	106	106	106	360	360
34	120	120	-	-	-	-	-	-	-	-	-	-	-	-
35	-	-	3	3	-	-	-	-	-	-	-	-	206	206
36	100	100	-	-	-	-	-	-	-	-	-	-	-	-
37	206	206	-	-	106	106	106	106	106	106	106	106	148	190
38	242	370	-	-	106	106	106	106	106	106	106	106	259	259
39	112	120	108	190	3	3	3	3	3	3	3	3	-	-
40	237	237	-	-	-	-	-	-	-	-	-	-	-	-
41	13	40	4	4	-	-	-	-	-	-	-	-	226	226
42 (link up)	120	120	209	209	-	-	-	-	-	-	-	-	133	150
43	-	-	161	210	206	206	206	206	206	206	206	206	3	3
44	282	290	3	3	-	-	-	-	-	-	-	-	-	-
45	101	190	159	210	184	184	184	184	184	184	184	184	-	-
46	-	-	106	106	-	-	-	-	-	-	-	-	209	209
47	106	106	-	-	106	106	106	106	106	106	106	106	106	106
48	228	290	33	40	228	228	228	228	228	228	228	228	228	290

Albany - Andrews Link Loss



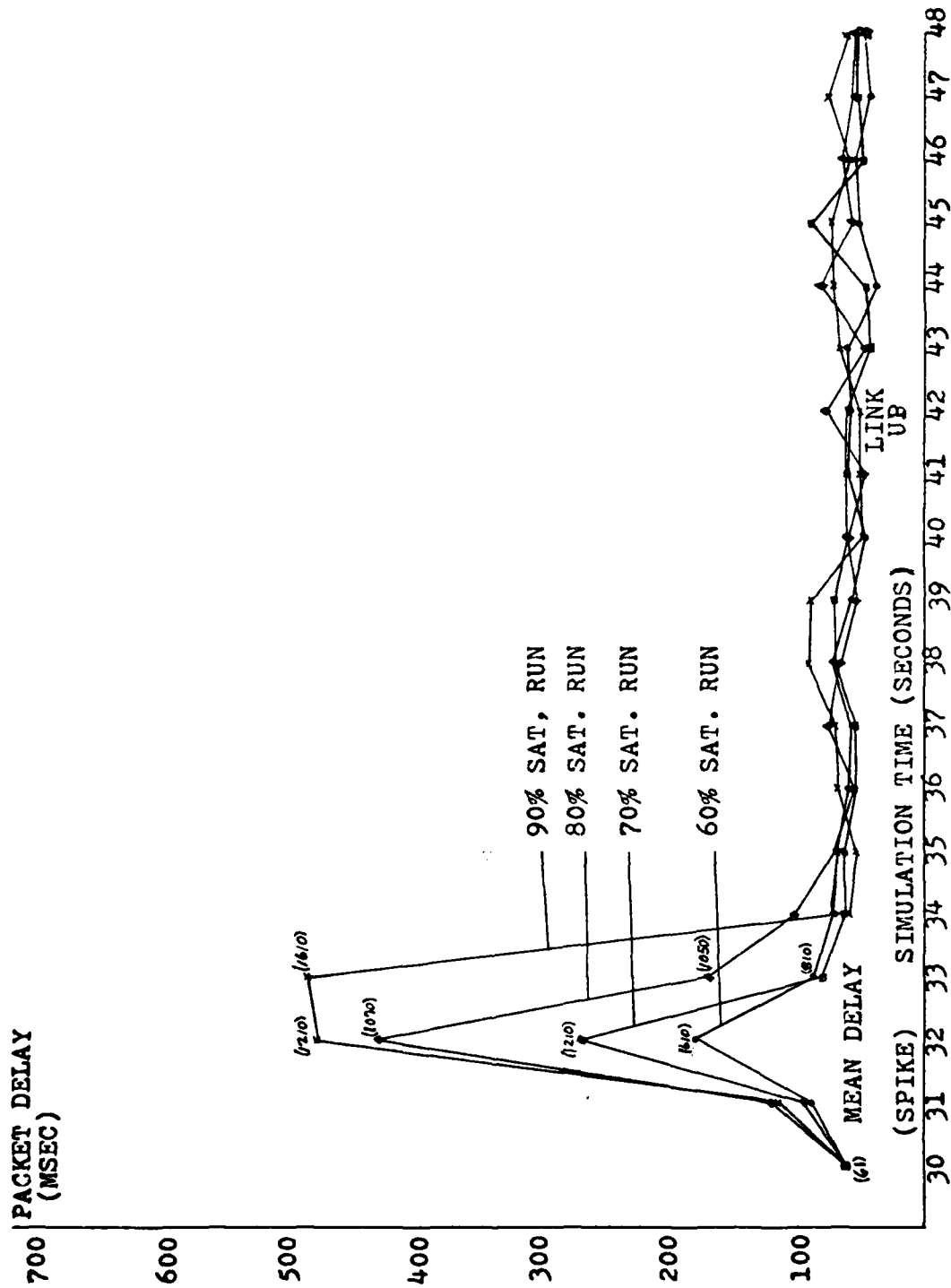


Fig H-5. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

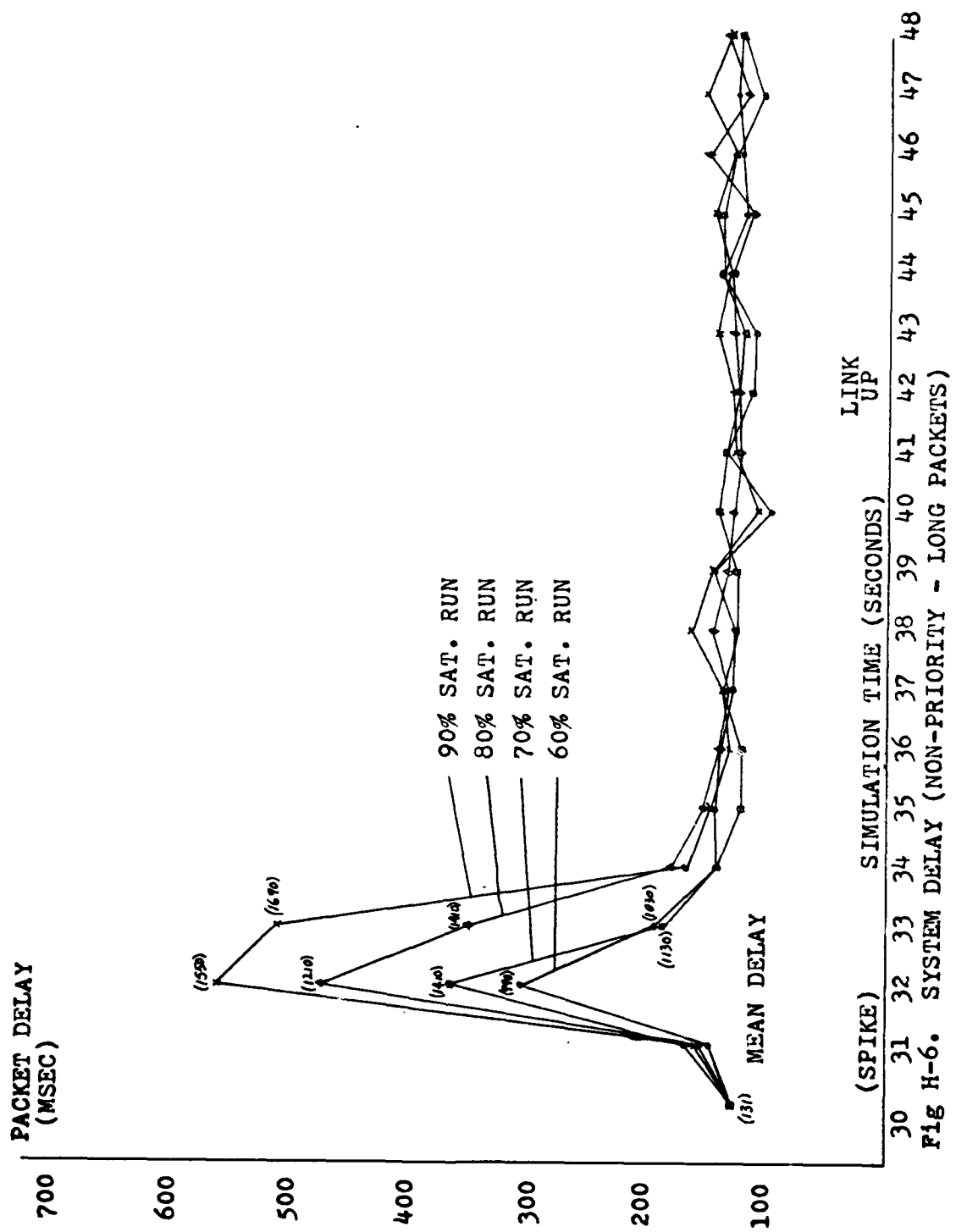


Fig H-6. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE H-III  
System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)				80% Sat.				90% Sat.			
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	62	230	62	230	62	230	62	230	62	230	62	230
31 (spike time)	186	186	55	90	46	90	46	90	85	110	85	110
32	68	110	113	120	-	-	-	-	61	61	61	61
33	-	-	33	33	3	3	3	3	-	-	-	-
34	-	-	-	-	-	-	-	-	-	-	-	-
35	4	4	-	-	-	-	-	-	97	130	97	130
36	33	33	89	90	78	78	78	78	-	-	-	-
37	38	70	33	33	52	110	52	110	-	-	-	-
38	-	-	-	-	33	33	33	33	-	-	-	-
39	-	-	-	-	-	-	-	-	-	-	-	-
40	-	-	3	3	-	-	-	-	-	-	-	-
41	-	-	-	-	-	-	-	-	3	3	3	3
42 (link up)	-	-	-	-	-	-	-	-	-	-	-	-
43	-	-	-	-	-	-	-	-	34	34	34	34
44	-	-	-	-	33	33	33	33	3	3	3	3
45	36	70	-	-	35	35	35	35	33	33	33	33
46	79	79	48	48	34	34	34	34	3	3	3	3
47	-	-	13	30	-	-	-	-	6	6	6	6
48	33	66	31	80	7	7	7	7	-	-	-	-

TABLE H-IV  
System Delay (priority - long packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Max	Run	Max	Mean	Max	Run	Max	Mean	Max	Run	Max	Mean	Max
30	135	310	310	310	135	310	310	310	135	310	310	310	135	310
31 (spike time)	160	270	270	110	56	110	110	390	169	390	250	250	126	250
32	185	290	290	-	-	-	210	330	155	210	330	330	158	330
33	-	-	86	150	5	5	290	290	147	290	290	290	147	290
34	106	106	3	3	106	106	106	106	-	-	-	-	-	-
35	89	110	-	-	-	-	-	-	-	-	-	-	-	-
36	209	209	-	-	55	110	110	3	3	3	3	3	3	3
37	-	-	-	-	106	106	106	106	106	106	106	106	-	-
38	106	106	-	-	107	190	190	250	250	250	250	250	250	250
39	-	-	129	270	181	181	181	-	-	-	-	-	-	-
40	209	209	149	270	-	-	-	106	106	106	106	106	106	106
41	158	210	-	-	263	263	263	263	263	263	263	263	-	-
42(link up)	-	-	209	209	55	110	110	170	170	170	170	170	92	170
43	3	3	106	106	-	-	-	-	-	-	-	-	-	-
44	62	110	118	130	58	110	110	310	146	310	310	310	146	310
45	76	230	169	230	209	209	209	3	3	3	3	3	3	3
46	66	130	3	3	-	-	-	5	5	5	5	5	5	5
47	-	-	47	47	16	16	16	-	-	-	-	-	-	-
48	106	106	3	3	-	-	-	40	40	40	40	40	31	40



Andrews - Hancock Link Loss

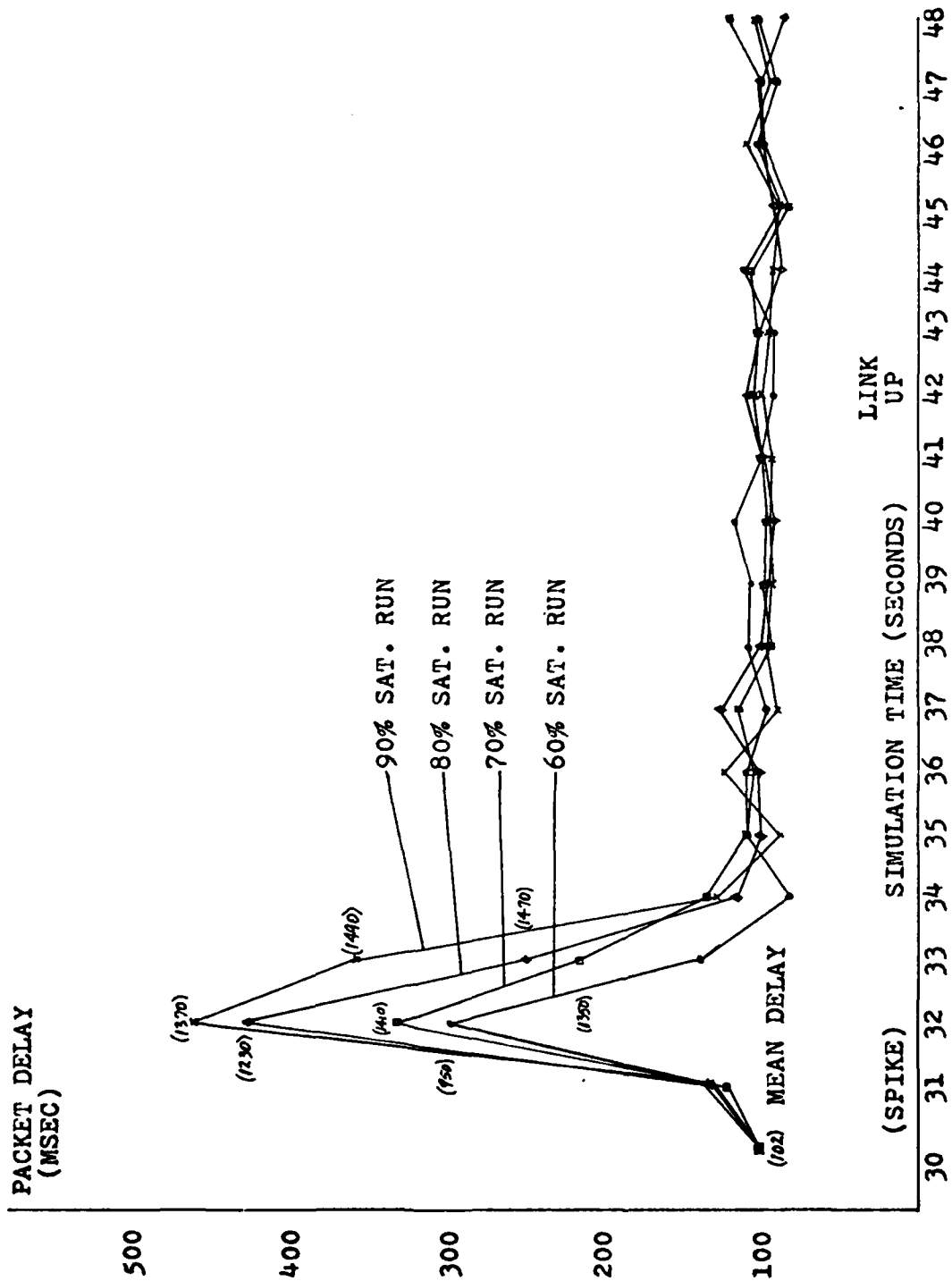


Fig H-8. SYSTEM DELAY (ALL PACKETS)

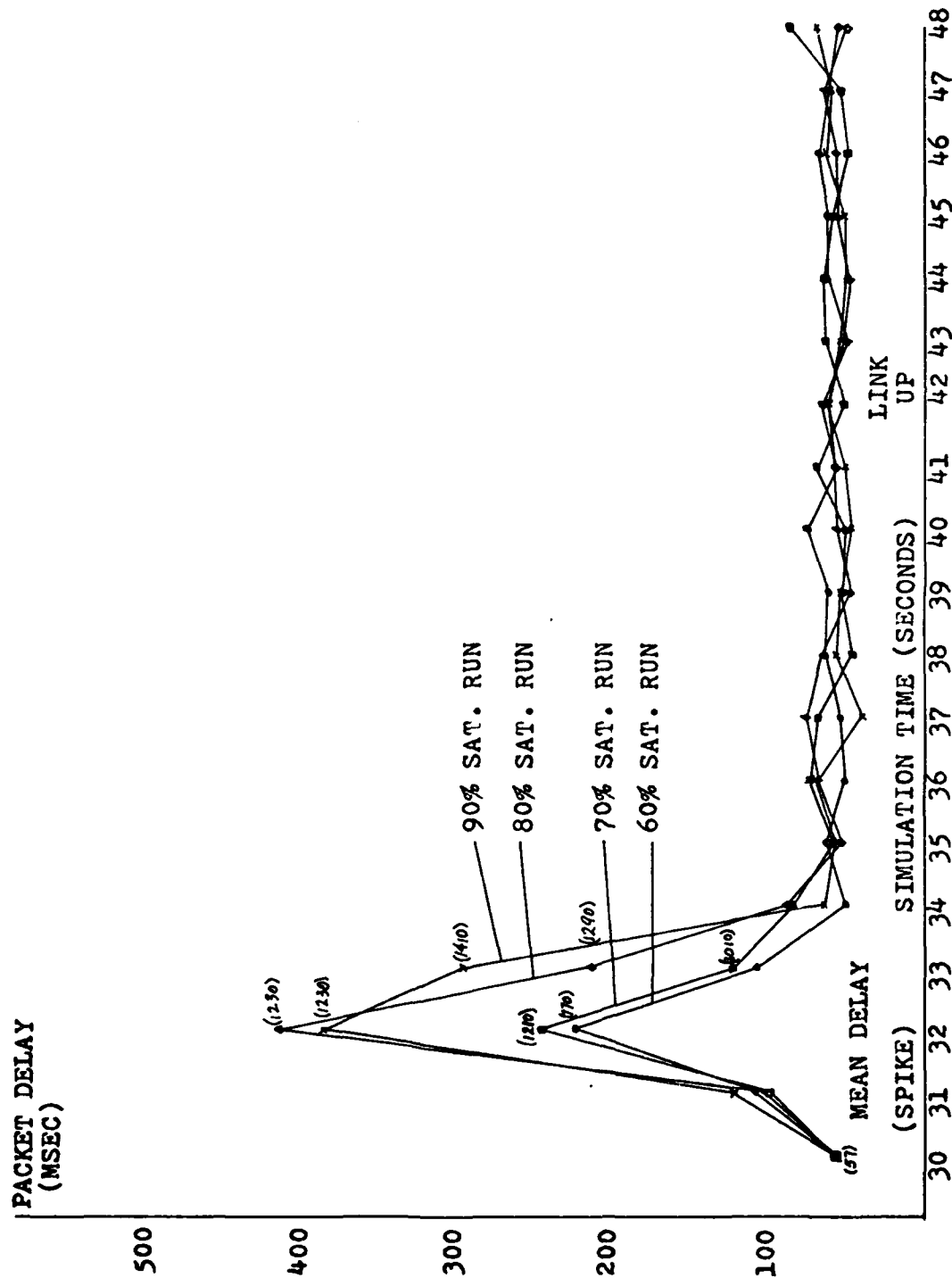


Fig H-9. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

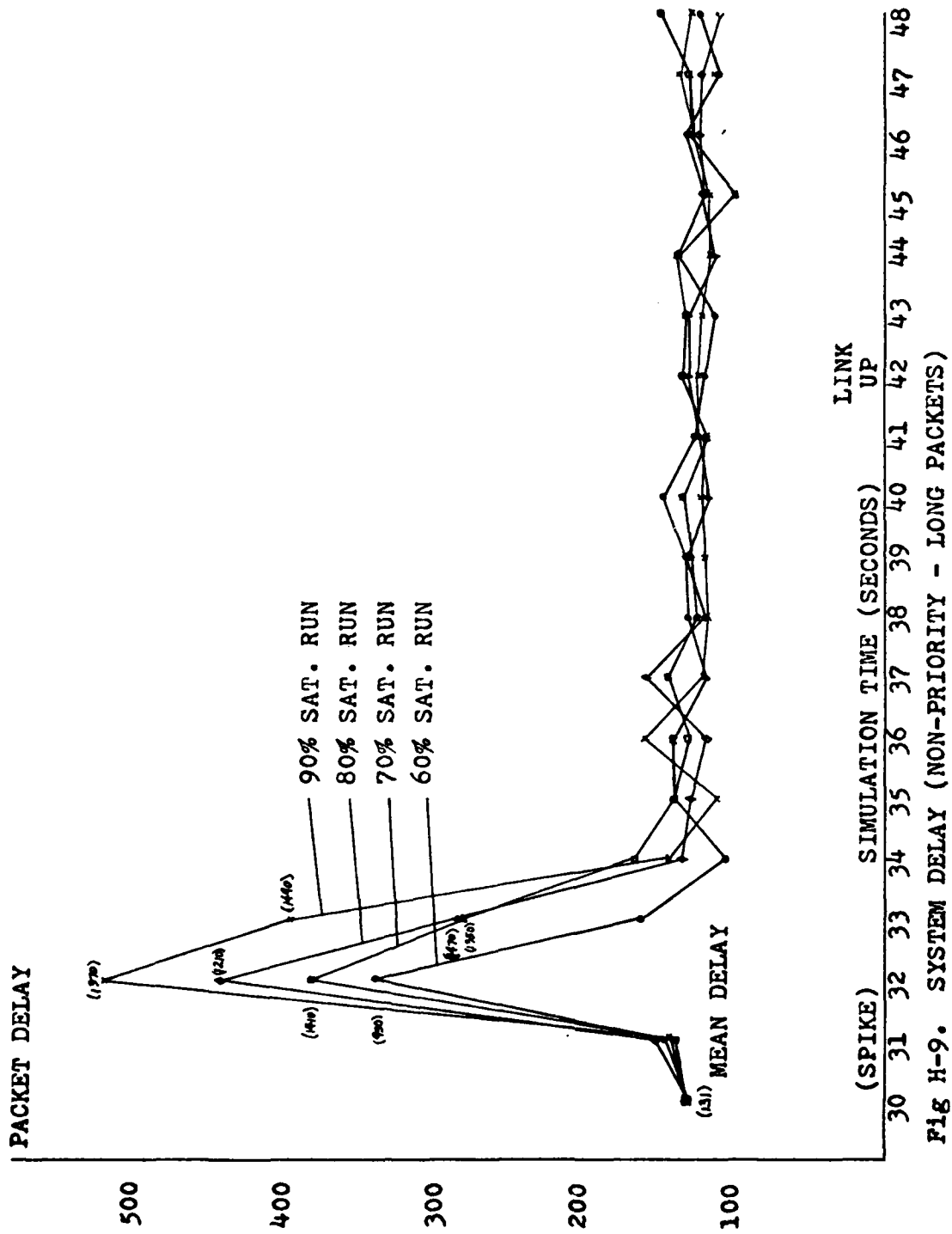


Fig H-9. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

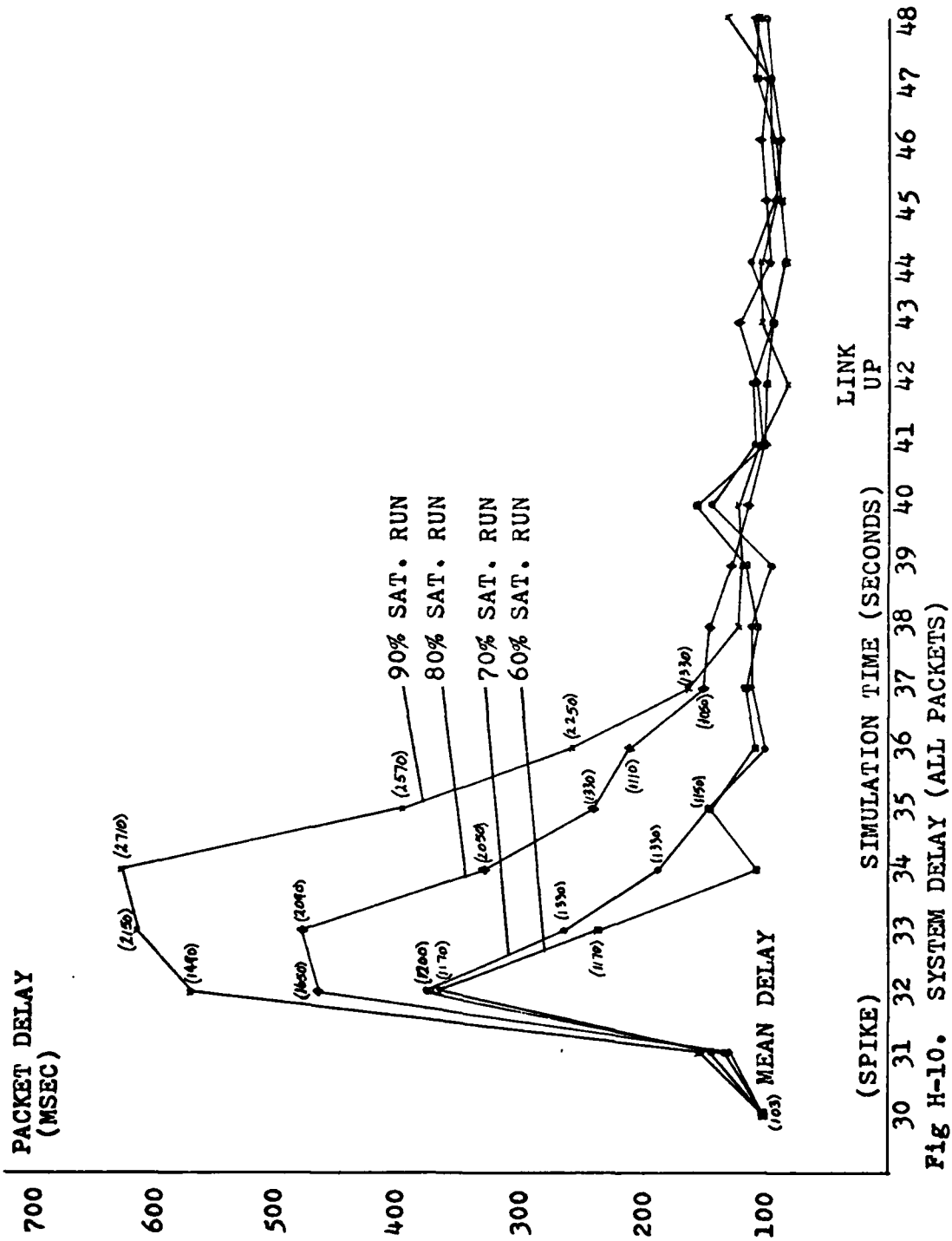
TABLE H-V  
System Delay (priority -short packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max
30	122	250	122	250	122	250	122	250	122	250	122	250	122	250
31 (spike time)	74	210	95	170	234	350	270	310	96	270	270	270	96	270
32	106	106	189	290	178	310	178	310	-	-	-	-	-	-
33	81	110	164	250	3	3	3	3	-	-	-	-	-	-
34	3	3	-	-	213	213	213	213	3	3	3	3	3	3
35	-	-	55	110	-	-	-	-	3	3	3	3	3	3
36	-	-	3	3	-	-	-	-	196	270	270	270	196	270
37	74	150	-	-	-	-	-	-	209	209	209	209	209	209
38	73	150	116	116	106	210	106	210	55	110	110	110	55	110
39	188	188	225	225	5	5	5	5	-	-	-	-	-	-
40	181	250	57	110	111	111	111	111	6	6	6	6	6	6
41	-	-	-	-	162	210	162	210	3	3	3	3	3	3
42 (link up)	-	-	107	107	106	106	106	106	197	197	197	197	197	197
43	130	130	-	-	200	210	200	210	-	-	-	-	-	-
44	-	-	106	106	193	250	193	250	106	106	106	106	106	106
45	-	-	149	190	-	-	-	-	-	-	-	-	-	-
46	107	110	208	310	-	-	-	-	-	-	-	-	-	-
47	-	-	-	-	-	-	-	-	38	50	50	50	38	50
48	302	302	-	-	-	-	-	-	3	3	3	3	3	3

TABLE H-VI  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)				90% Sat.			
	60% Sat. Mean	70% Sat. Mean	80% Sat. Mean	Run Max	Mean	Run Max	Mean	Run Max
30	60	60	60	230	60	230	60	230
31 (spike time)	161	3	111	3	109	230	109	130
32	3	128	-	128	75	-	75	75
33	-	-	52	-	-	70	-	-
34	-	-	5	-	55	5	55	70
35	-	-	21	-	49	30	49	49
36	-	33	-	33	65	-	65	65
37	113	-	3	-	-	3	-	-
38	33	-	-	-	43	-	43	70
39	3	-	148	-	-	230	-	-
40	69	-	-	-	-	-	-	-
41	-	-	-	-	63	-	63	63
42 (link up)	-	-	45	-	-	-	-	-
43	63	-	3	-	-	45	-	-
44	-	-	-	-	-	3	-	-
45	64	-	84	-	-	-	-	-
46	33	-	-	-	24	130	24	70
47	20	-	3	-	33	-	33	33
48	-	3	-	3	-	3	-	-
	-	-	-	-	176	-	176	176

Andrews - Ft. Detrick Link Loss





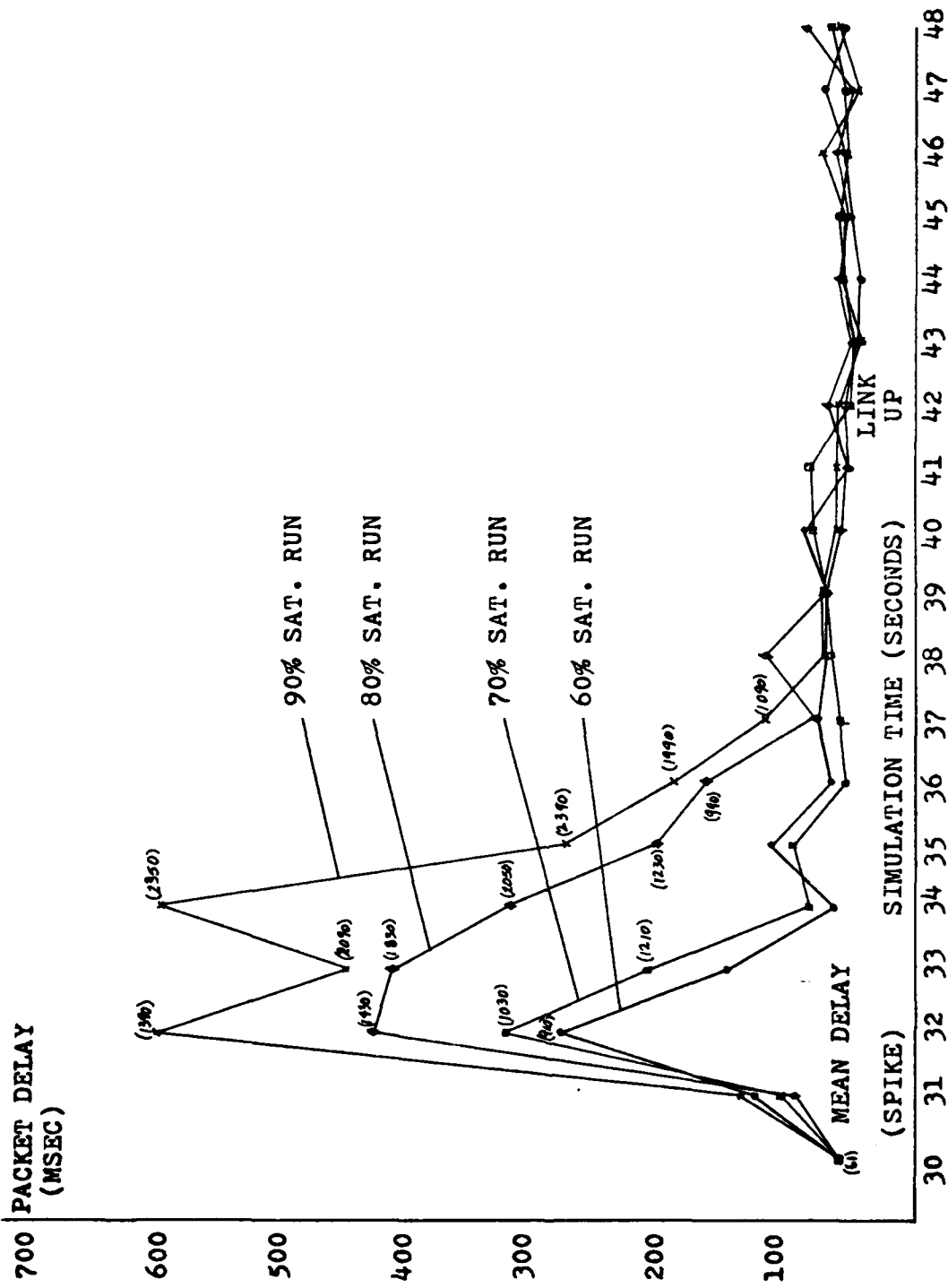


Fig H-11. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)



TABLE H-VII  
System Delay (priority - short packets)

Simulation Time (seconds)	60% Sat.			70% Sat.			80% Sat.			90% Sat.		
	Mean	Run Max	Delay Times (milliseconds)	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean
30	64	230	64	230	64	230	64	230	64	230	64	230
31 (spike time)	96	130	62	110	3	3	3	3	3	-	-	-
32	82	82	69	210	-	-	-	-	-	59	59	59
33	-	-	33	33	65	65	65	65	65	-	-	-
34	-	-	-	-	-	-	-	-	-	80	230	230
35	-	-	-	-	3	3	3	3	3	-	-	-
36	57	60	33	33	48	48	48	48	48	-	-	-
37	-	-	-	-	47	47	47	47	47	105	130	130
38	32	50	6	6	41	41	41	41	41	-	-	-
39	123	123	-	-	-	-	-	-	-	-	-	-
40	18	30	111	111	3	3	3	3	3	-	-	-
41	-	-	-	-	-	-	-	-	-	35	35	35
42 (link up)	37	37	-	-	-	-	-	-	-	-	-	-
43	-	-	43	43	88	88	88	88	88	-	-	-
44	3	3	-	-	-	-	-	-	-	63	63	63
45	-	-	63	63	-	-	-	-	-	-	-	-
46	-	-	33	33	-	-	-	-	-	-	-	-
47	-	-	3	3	48	48	48	48	48	63	63	63
48	3	3	54	54	-	-	-	-	-	-	-	-

TABLE H-VIII  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)						90% Sat.	
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	80% Sat. Mean	Run Max	Mean	Run Max
30	122	390	122	390	122	390	122	390
31 (spike time)	107	310	153	190	141	370	48	110
32	106	106	-	-	-	-	294	294
33	3	3	4	10	118	230	-	-
34	106	106	-	-	101	170	-	-
35	106	106	205	205	106	210	58	110
36	-	-	-	-	-	-	130	130
37	131	131	-	-	83	130	196	310
38	-	-	3	3	133	150	3	3
39	108	108	55	110	3	3	-	-
40	193	193	133	270	106	106	-	-
41	-	-	3	3	218	218	106	106
42 (link up)	3	3	-	-	222	250	-	-
43	-	-	209	209	106	106	-	-
44	-	-	-	-	210	210	106	106
45	3	3	209	209	6	10	200	200
46	3	3	4	4	-	-	-	-
47	200	200	209	209	-	-	108	108
48	106	106	3	3	42	110	-	-

Andrews - Tinker Link Loss

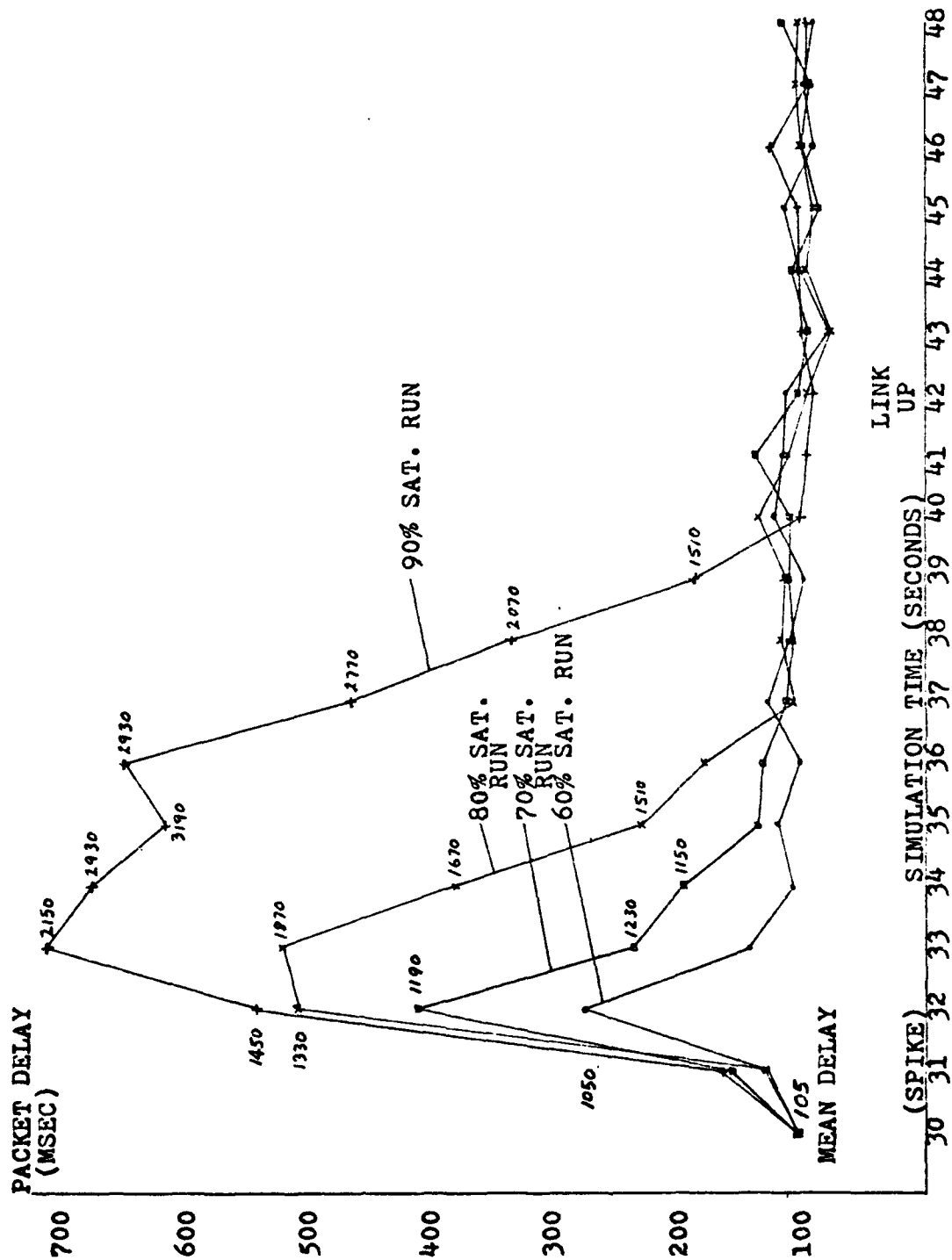


Fig H-13. SYSTEM DELAY (ALL PACKETS)

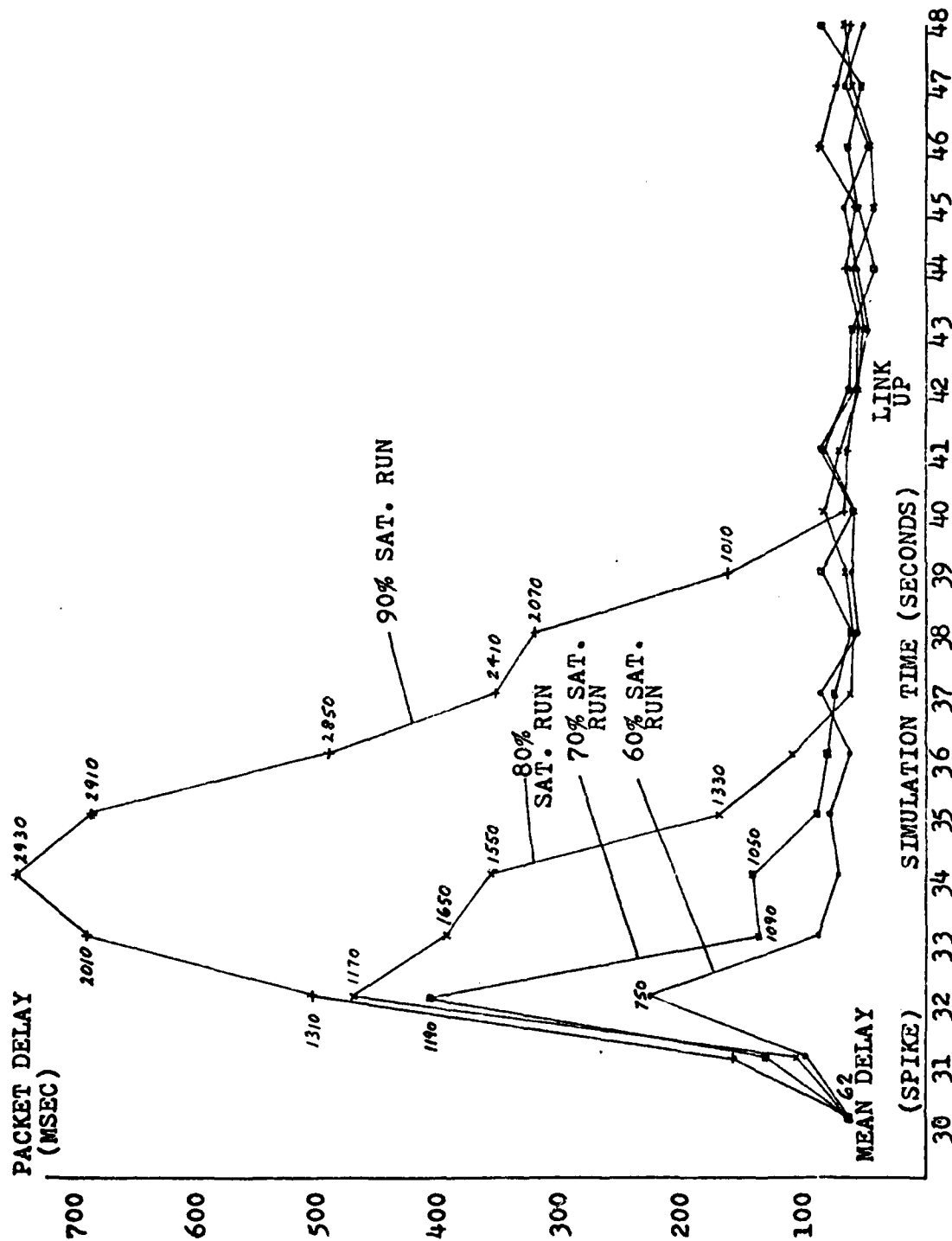
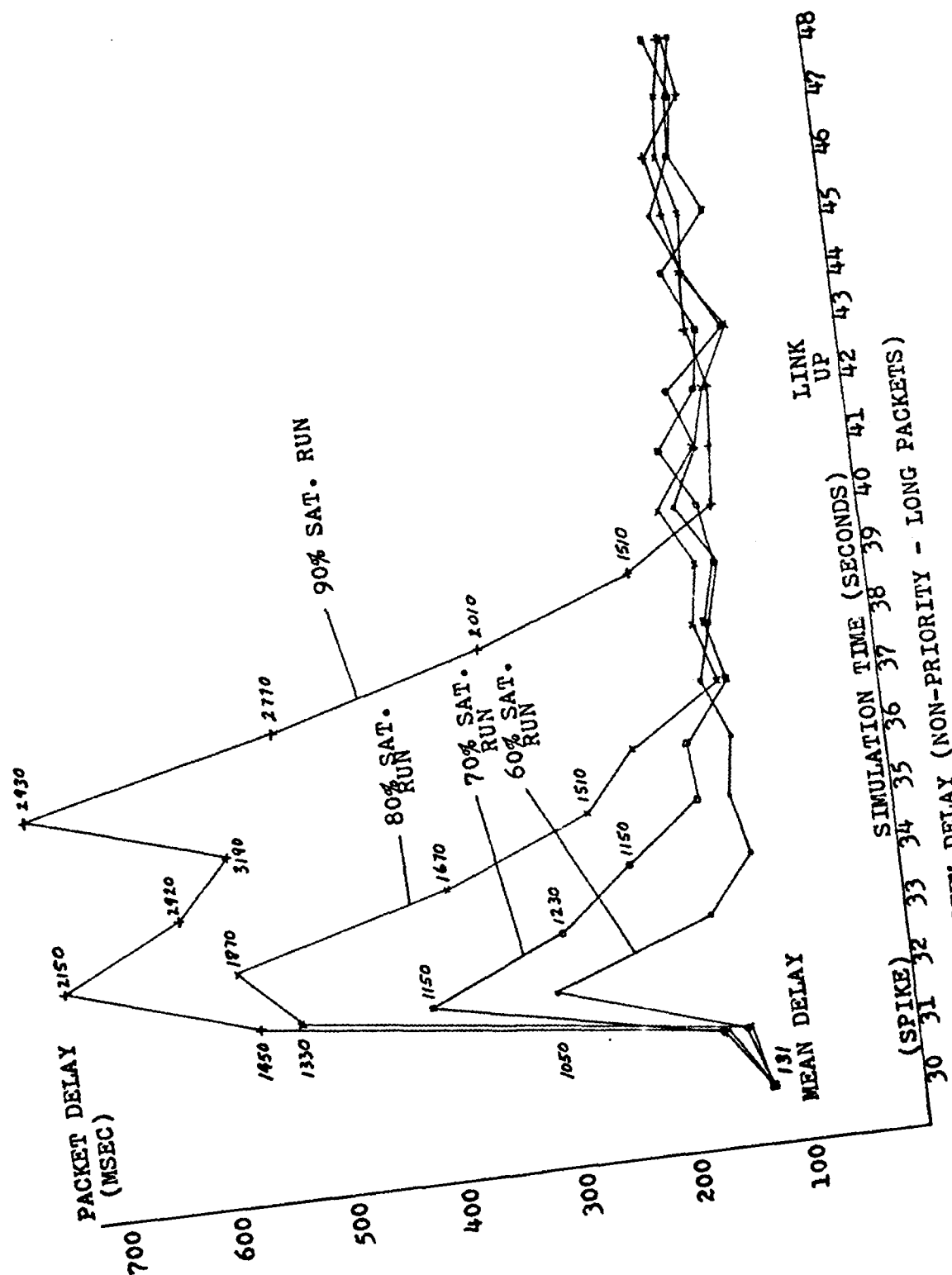


Fig H-14. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)



LINK  
UP  
SIMULATION TIME (SECONDS)

(SPIKE)

30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

Fig H-15. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)



TABLE H-IX  
System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)				80% Sat.				90% Sat.			
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	62	230	62	230	62	230	62	230	62	230	62	230
31	97	130	53	90	3	3	3	3	112	112	112	112
32 (spike time)	-	-	-	-	-	-	-	-	-	-	-	-
33	-	-	141	141	3	3	3	3	7	7	7	7
34	63	63	26	50	-	-	-	-	3	3	3	3
35	-	-	5	10	6	6	6	6	-	-	-	-
36	47	47	-	-	-	-	-	-	-	-	-	-
37	3	3	3	3	-	-	-	-	-	-	-	-
38	3	3	3	3	-	-	-	-	-	-	-	-
39	-	-	-	-	-	-	-	-	-	-	-	-
40	-	-	-	-	-	-	-	-	121	121	121	121
41	-	-	-	-	-	-	-	-	-	-	-	-
42 (link up)	-	-	-	-	-	-	-	-	63	63	63	63
43	33	33	34	34	-	-	-	-	-	-	-	-
44	-	-	-	-	-	-	-	-	33	33	33	33
45	-	-	-	-	33	33	33	33	3	3	3	3
46	3	3	3	3	3	3	3	3	124	124	124	124
47	74	74	47	47	3	3	3	3	-	-	-	-
48	67	67	-	-	-	-	-	-	20	20	20	20

TABLE H-X  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)						90% Sat.		Run	
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	80% Sat. Mean	Run Max	Mean	Max	Mean	Max
30	127	350	127	350	127	350	127	350	127	350
31	202	230	139	170	96	210	84	170	84	170
32 (spike time)	-	-	251	390	455	455	180	350	180	350
33	134	150	148	290	200	210	316	316	316	316
34	210	210	116	230	-	-	209	209	209	209
35	257	310	-	-	-	-	6	6	6	6
36	-	-	3	10	-	-	175	210	175	210
37	-	-	209	209	121	230	-	-	-	-
38	160	210	-	-	-	-	-	-	-	-
39	3	3	132	250	133	133	83	130	83	130
40	263	263	68	130	182	310	-	-	-	-
41	-	-	106	106	267	310	38	110	38	110
42 (link up)	74	150	144	144	-	-	3	3	3	3
43	-	-	108	108	56	110	209	209	209	209
44	118	118	209	209	7	7	-	-	-	-
45	106	106	209	210	215	215	-	-	-	-
46	199	199	-	-	-	-	106	106	106	106
47	209	209	-	-	158	210	107	107	107	107
48	-	-	-	-	-	-	3	3	3	3

Gentile - Ft. Detrick Link Loss

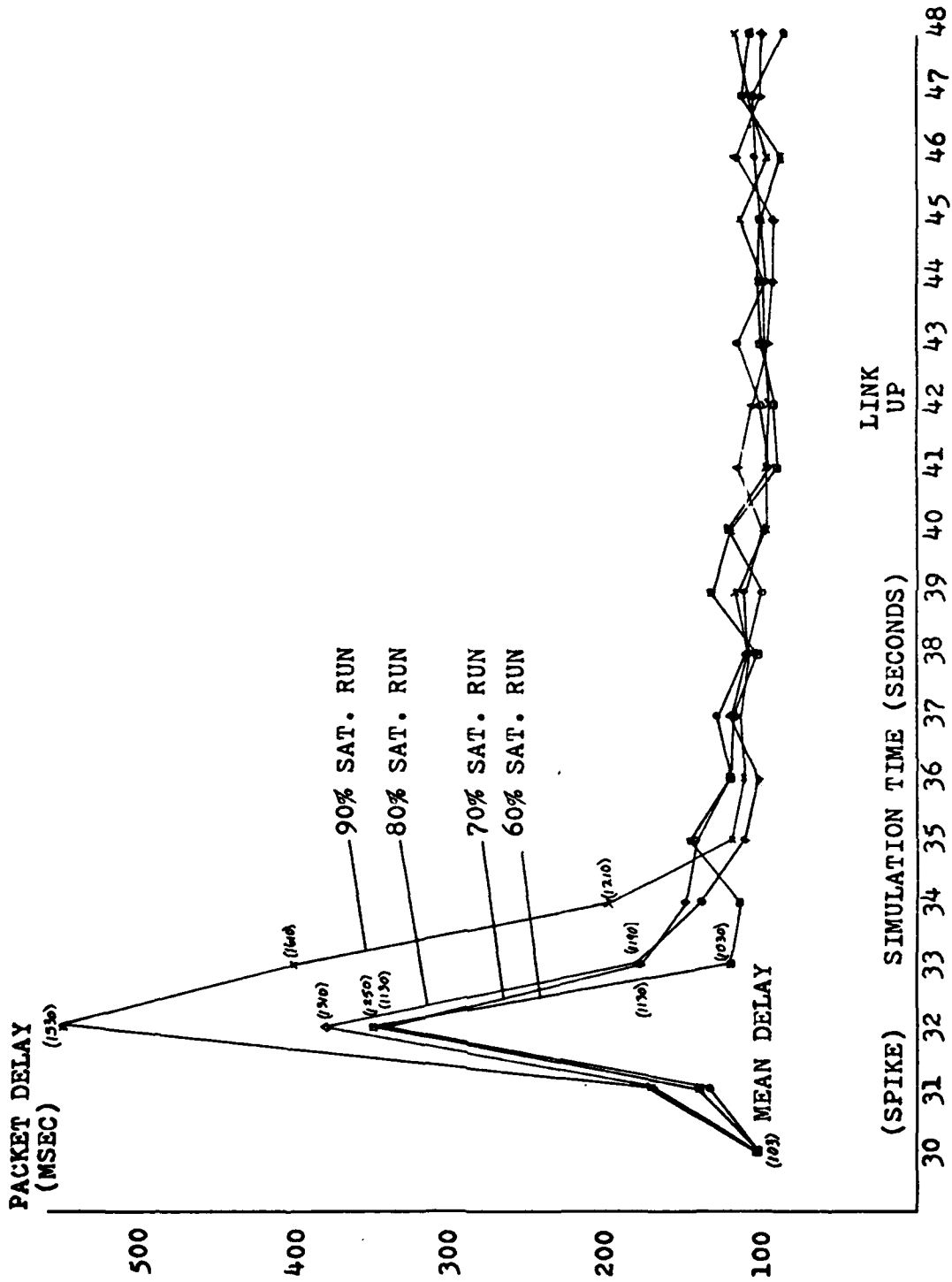


Fig H-16. SYSTEM DELAY (ALL PACKETS)

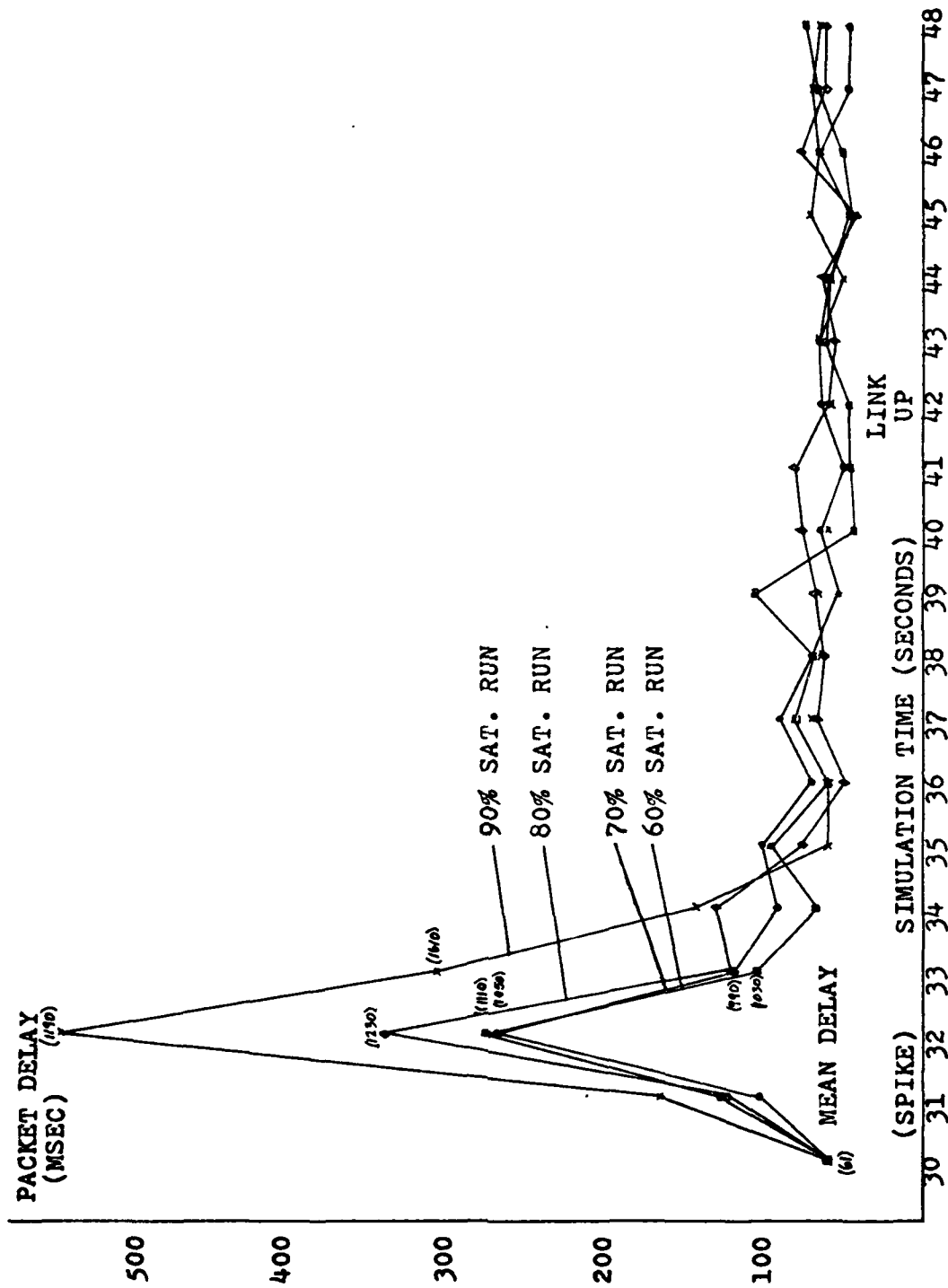


Fig H-17. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

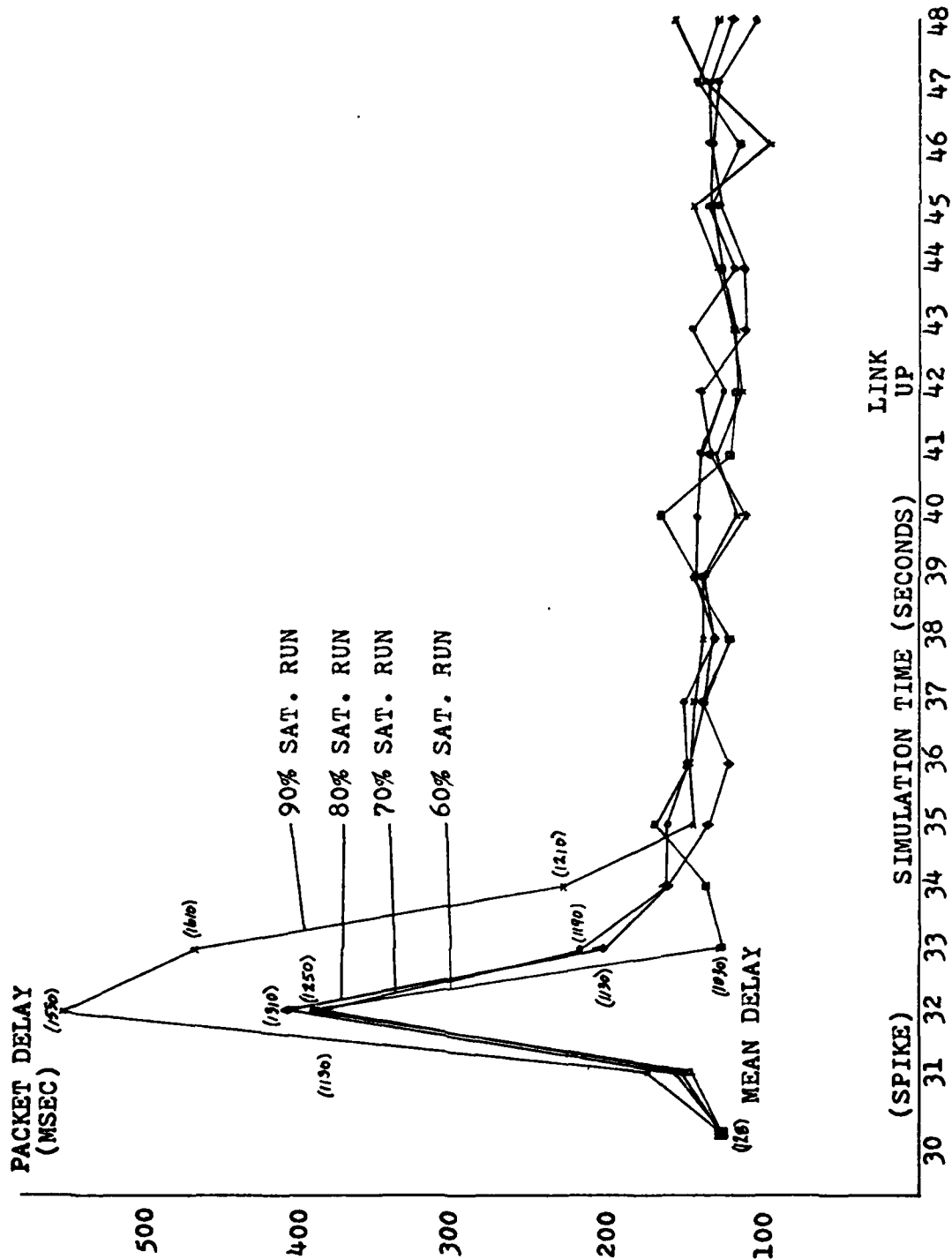


Fig H-18. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE H-XI  
System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)									
	60% Sat.		70% Sat.		80% Sat.		90% Sat.		Run	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
30	64	230	64	230	64	230	64	230	64	230
31 (spike time)	61	110	99	150	42	110	190	210		
32	-	-	-	-	-	-	-	-	-	-
33	3	3	-	-	-	-	36	36		
34	-	-	-	-	-	-	3	3		
35	-	-	35	35	35	70	163	163		
36	3	3	-	-	-	-	-	-		
37	48	70	117	117	68	68	-	-		
38	80	80	3	3	-	-	4	4		
39	-	-	-	-	39	70	-	-		
40	35	35	33	33	-	-	-	-		
41	-	-	-	-	-	-	-	-		
42 (link up)	44	44	-	-	139	139	108	108		
43	-	-	64	64	-	-	-	-		
44	-	-	-	-	33	33	33	33		
45	-	-	-	-	-	-	34	34		
46	74	110	18	30	3	3	118	118		
47	35	70	-	-	3	3	3	3		
48	-	-	-	-	-	-	-	-		

TABLE H-XII  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times					
	60% Sat.		70% Sat.		80% Sat.	
	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	122	350	122	350	122	350
31 (spike time)	156	250	60	110	86	170
32	-	-	-	-	172	172
33	170	170	-	-	223	223
34	66	250	82	130	76	230
35	255	255	-	-	-	-
36	286	330	55	110	-	-
37	106	106	196	196	-	-
38	3	3	183	183	-	-
39	-	-	3	3	192	270
40	158	210	43	230	3	3
41	3	3	3	3	-	-
42 (link up)	55	110	-	-	-	-
43	-	-	144	150	3	3
44	-	-	-	-	237	270
45	209	290	-	-	72	350
46	108	210	3	3	-	-
47	106	106	226	350	-	-
48	-	-	-	-	-	-



Ft. Detrick - Tinker Link Loss



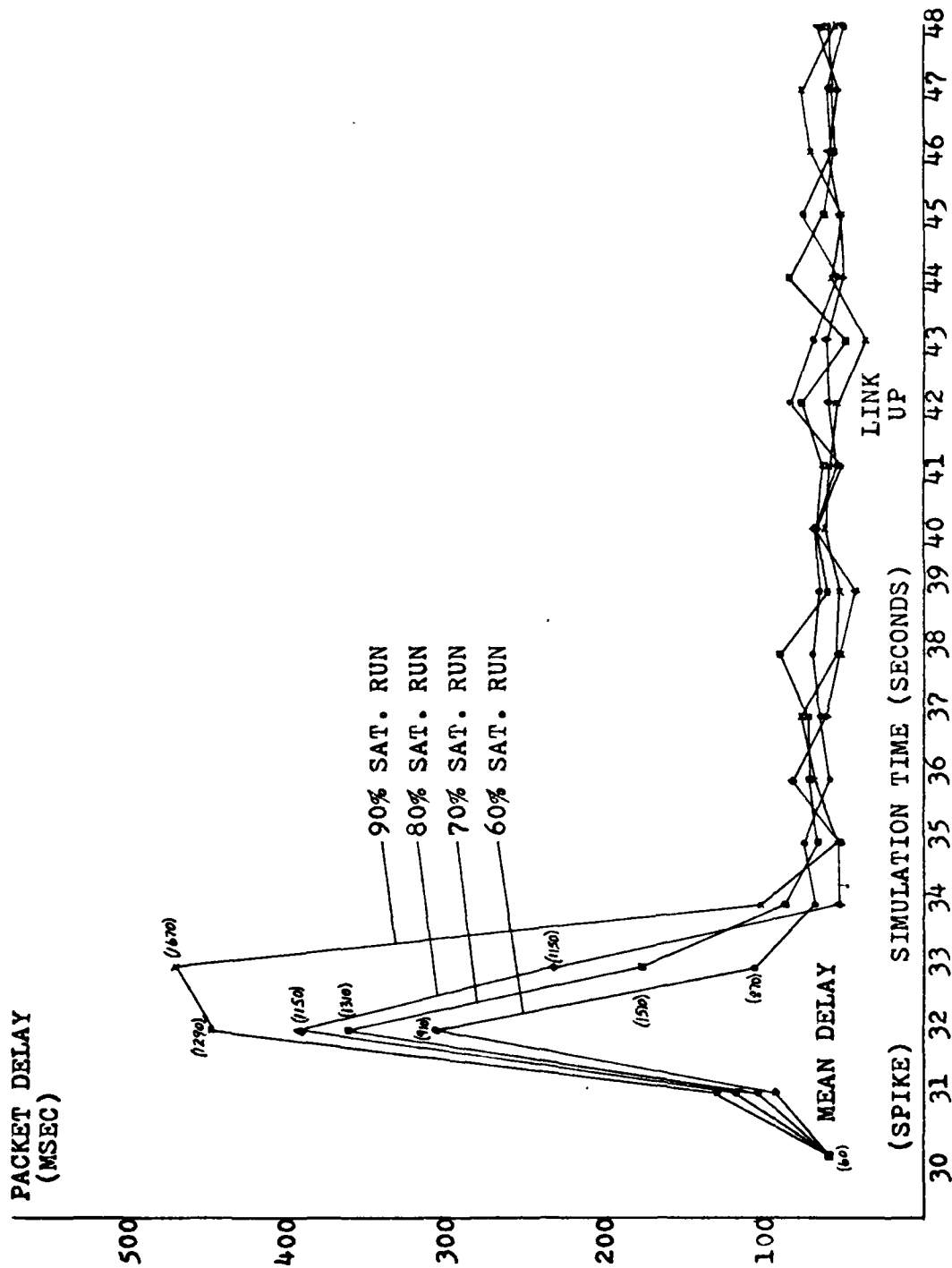


Fig H-20. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

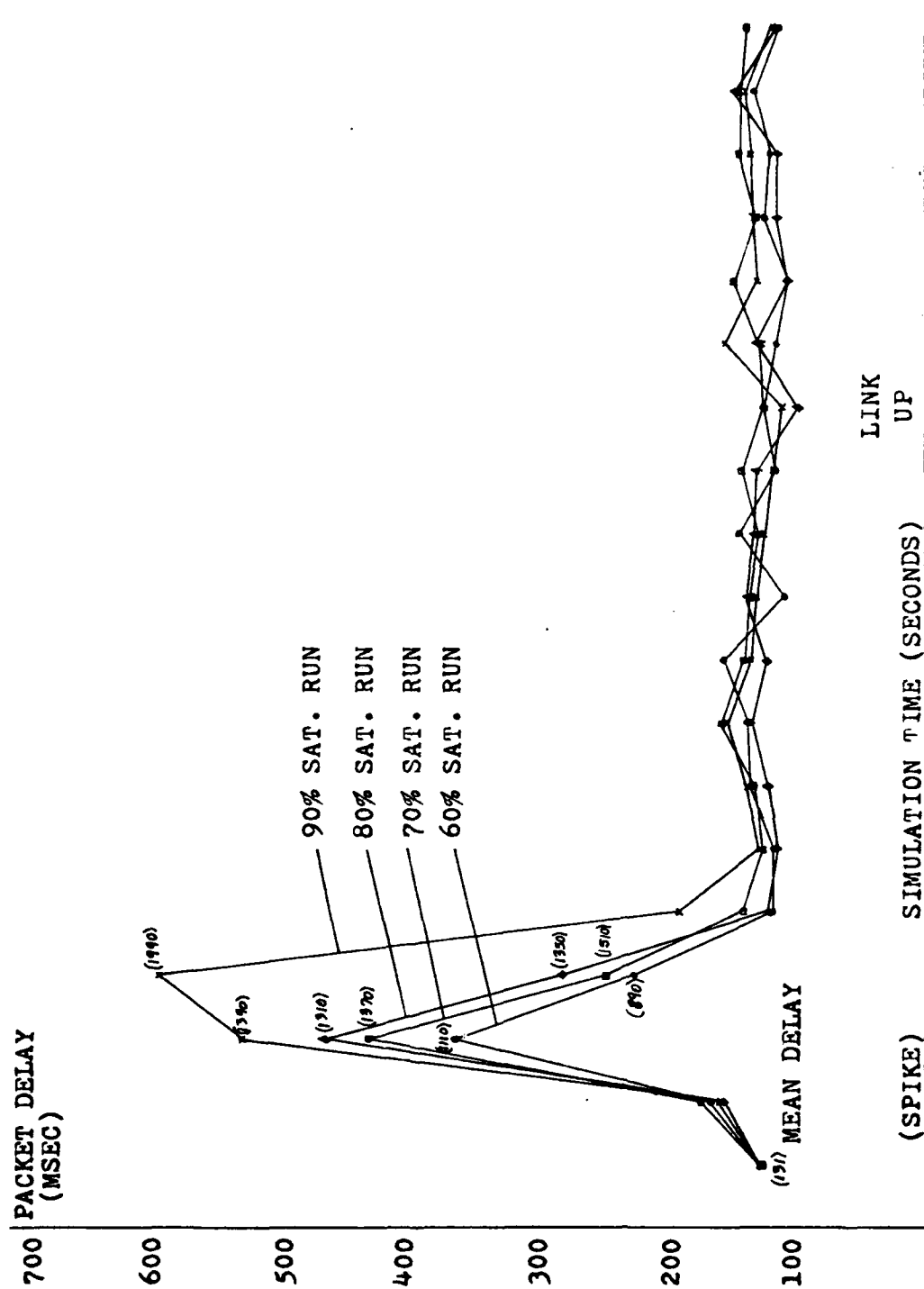


Fig H-21. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE H-XIII  
System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)				Delay Times (milliseconds)				Delay Times (milliseconds)			
	60% Sat. Mean	70% Sat. Mean	Run Max	Run Max	80% Sat. Mean	90% Sat. Mean	Run Max	Run Max	60% Sat. Mean	70% Sat. Mean	Run Max	Run Max
30	59	59	230	230	59	59	230	230	59	59	230	230
31 (spike time)	72	-	170	-	31	3	70	3	-	-	-	-
32	-	169	-	169	-	-	-	-	-	-	-	-
33	38	-	38	-	-	92	-	92	-	-	-	-
34	-	63	-	63	-	37	-	37	-	-	-	-
35	78	37	78	37	33	-	33	-	-	-	-	-
36	18	162	30	162	-	-	-	-	-	-	-	-
37	66	3	66	3	-	-	-	-	-	-	-	-
38	-	33	-	33	-	53	-	53	-	-	-	-
39	-	-	-	-	-	58	-	58	-	-	-	-
40	-	-	-	-	-	-	-	-	-	-	-	-
41	33	-	33	-	-	38	-	38	-	-	-	-
42 (link up)	-	33	-	33	-	-	-	-	-	-	-	-
43	3	-	3	-	-	-	-	-	-	-	-	-
44	-	-	-	-	4	92	4	92	-	-	-	-
45	3	3	3	3	-	-	-	-	-	-	-	-
46	-	105	-	105	-	37	-	37	-	-	-	-
47	70	63	70	63	-	-	-	-	-	-	-	-
48	33	-	33	-	-	-	-	-	-	-	-	-

TABLE H-XIV  
System Delay (priority - long packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max
30	105	230	105	230	105	230	105	230	105	230	105	230	105	230
31 (spike time)	127	150	133	270	133	270	128	250	128	250	160	370	160	370
32	201	390	279	370	279	370	113	210	113	210	142	250	142	250
33	3	3	193	210	3	210	3	3	3	3	7	7	7	7
34	294	294	55	110	55	110	-	-	-	-	-	-	-	-
35	3	3	211	211	211	211	-	-	-	-	106	106	106	106
36	-	-	185	185	185	185	106	106	106	106	82	150	82	150
37	229	229	115	130	115	130	95	170	95	170	109	150	109	150
38	-	-	55	110	55	110	209	209	209	209	55	110	55	110
39	174	174	130	130	130	130	160	210	160	210	118	130	118	130
40	106	106	-	-	-	-	210	210	210	210	-	-	-	-
41	3	3	-	-	-	-	6	6	6	6	-	-	-	-
42 (link up)	-	-	225	225	225	225	-	-	-	-	6	6	6	6
43	219	219	-	-	-	-	-	-	-	-	-	-	-	-
44	-	-	159	159	159	159	6	6	6	6	57	110	57	110
45	-	-	124	190	124	190	106	106	106	106	121	121	121	121
46	140	190	-	-	-	-	106	106	106	106	122	122	122	122
47	-	-	-	-	-	-	-	-	-	-	-	-	-	-
48	-	-	-	-	-	-	-	-	-	-	109	109	109	109

Gentile - Tinker Link Loss

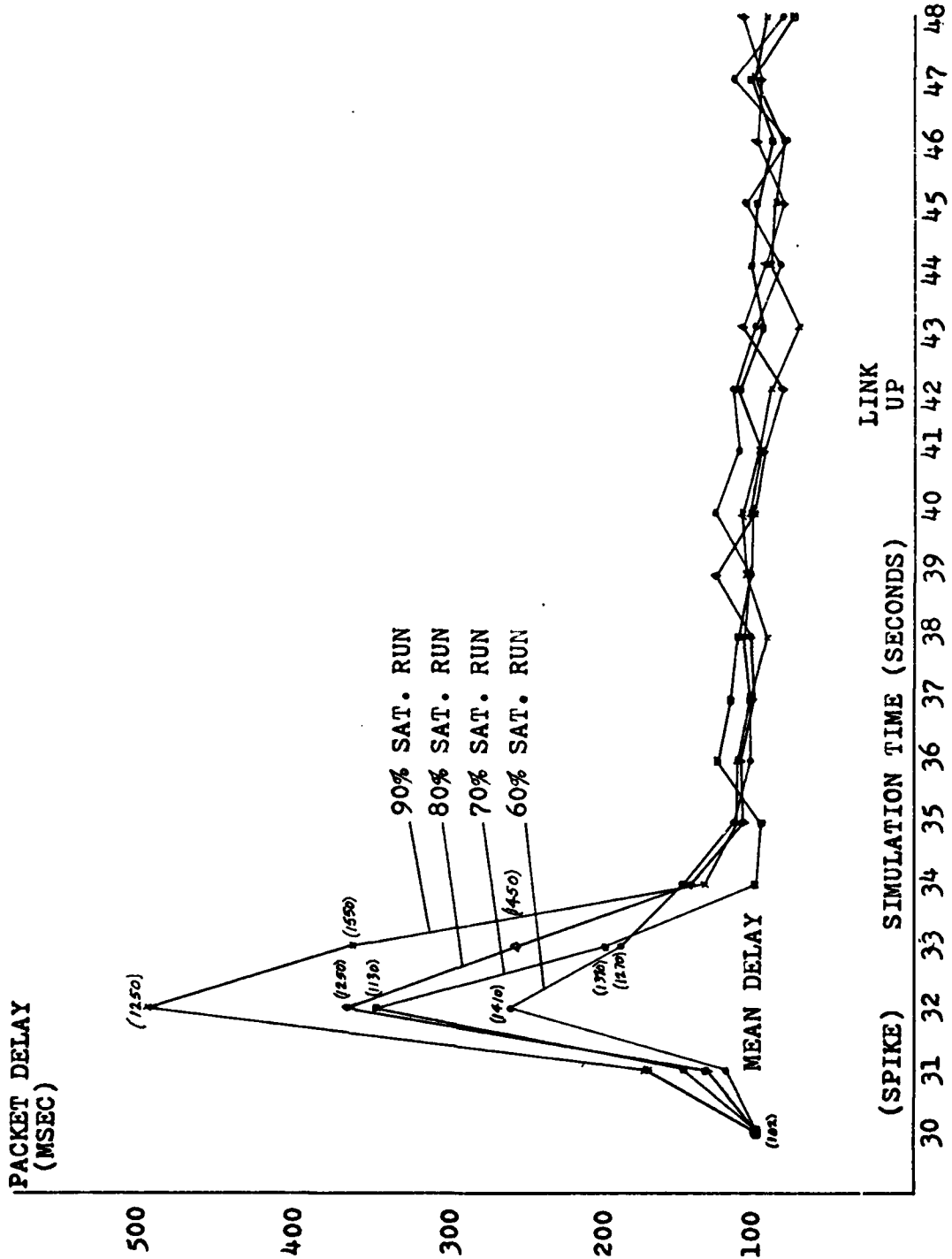


Fig H-22. SYSTEM DELAY (ALL PACKETS)



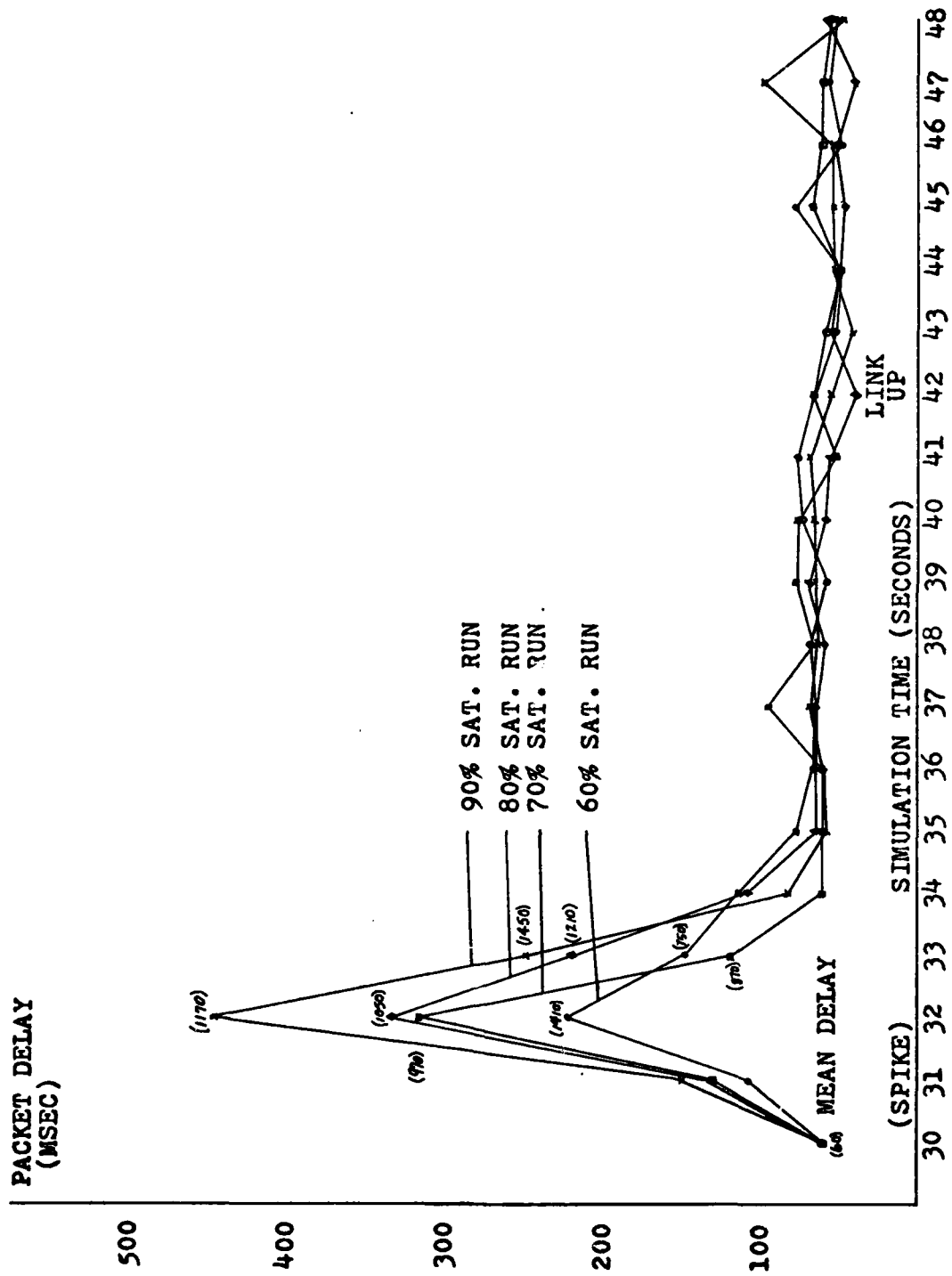


Fig H-23. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

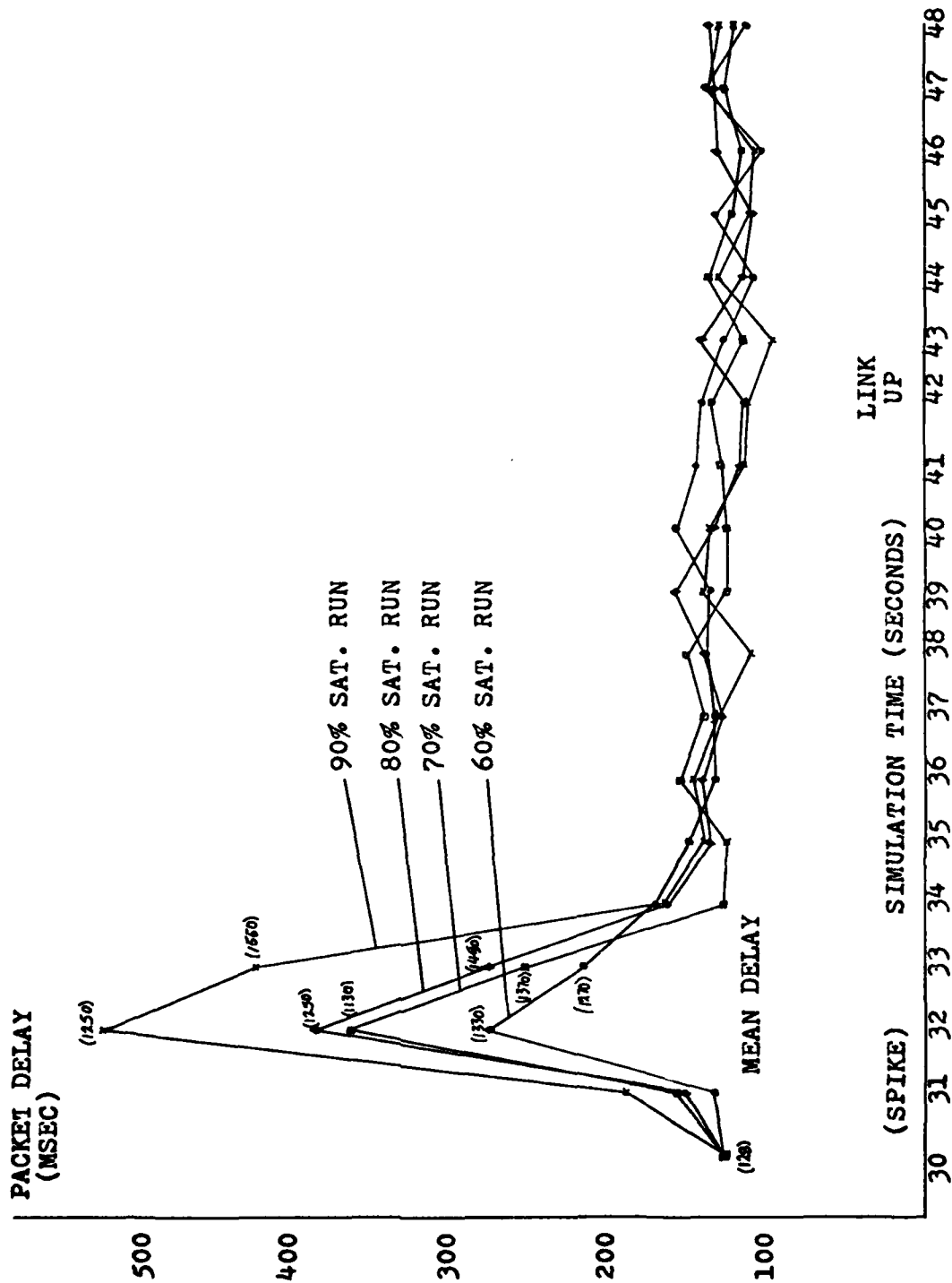


Fig H-24. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE H-XV  
System Delay (priority - short packets)

Simulation Time (seconds)	60% Sat.			70% Sat.			80% Sat.			90% Sat.	
	Mean	Run Max	Delay Times (milliseconds)	Mean	Run Max	Mean	Run Max	Mean	Run Max	Mean	Run Max
30	65	230	65	65	230	65	230	65	230	65	230
31 (spike time)	66	110	75	75	90	45	130	45	130	80	150
32	120	120	8	8	8	-	-	-	-	75	75
33	44	44	-	-	-	-	-	-	-	-	-
34	45	50	-	-	-	-	-	-	-	-	-
35	63	63	-	-	-	-	-	-	-	63	63
36	-	-	97	97	190	-	-	-	-	24	70
37	56	130	3	3	3	48	70	-	-	-	-
38	33	33	-	-	-	-	-	-	-	-	-
39	-	-	33	33	33	-	-	-	-	3	3
40	3	3	-	-	-	-	-	-	-	-	-
41	-	-	-	-	-	3	3	-	-	66	130
42 (link up)	-	-	-	-	-	-	-	-	-	-	-
43	-	-	19	19	30	-	-	-	-	-	-
44	-	-	-	-	-	-	-	-	-	3	3
45	33	33	57	57	57	-	-	-	-	84	130
46	3	3	-	-	-	3	3	-	-	18	30
47	3	3	33	33	33	-	-	-	-	20	30
48	-	-	63	63	63	-	-	-	-	17	17

TABLE H-XVI  
System Delay (priority - long packets)

Simulation Time (seconds)	System Delay				Delay Times (milliseconds)				80% Sat.		90% Sat.	
	60% Sat.		70% Sat.		Run		Run		Mean	Max	Mean	Max
30	121	350	121	350	121	350	121	350	121	350	121	350
31 (spike time)	109	190	113	330	113	330	130	170	125	310	125	310
32	-	-	-	-	-	-	236	250	256	290	256	290
33	111	111	62	110	110	110	147	210	7	7	7	7
34	224	350	106	106	106	106	209	209	3	3	3	3
35	107	107	106	106	106	106	3	3	107	107	107	107
36	3	3	-	-	-	-	232	232	11	11	11	11
37	106	106	106	106	106	106	137	170	141	141	141	141
38	87	150	166	210	210	210	73	110	84	130	84	130
39	106	106	-	-	-	-	190	210	3	3	3	3
40	140	230	80	110	110	110	85	250	158	210	158	210
41	106	106	3	3	3	3	208	210	-	-	-	-
42 (link up)	-	-	203	203	203	203	106	106	-	-	-	-
43	211	211	-	-	-	-	-	-	113	113	113	113
44	-	-	96	190	190	190	297	297	95	170	95	170
45	188	210	111	210	210	210	114	230	106	106	106	106
46	106	106	11	11	11	11	-	-	-	-	-	-
47	-	-	209	209	209	209	-	-	-	-	-	-
48	-	-	163	190	190	190	106	106	8	8	8	8

Appendix I  
Graphical and Tabular Delay Results  
From the Nodal Loss Runs

Loss of Ft. Detrick



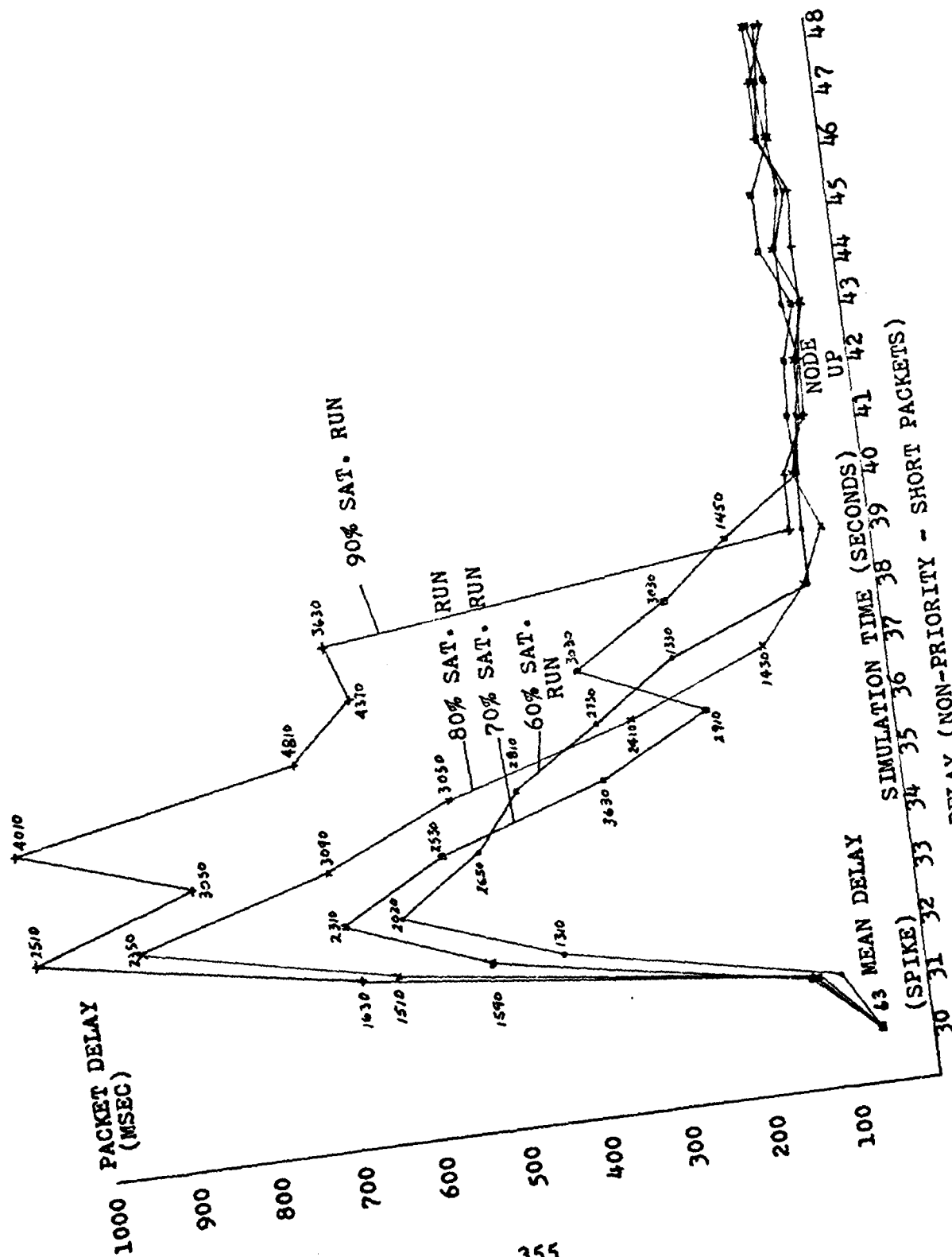


Fig I-2. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)



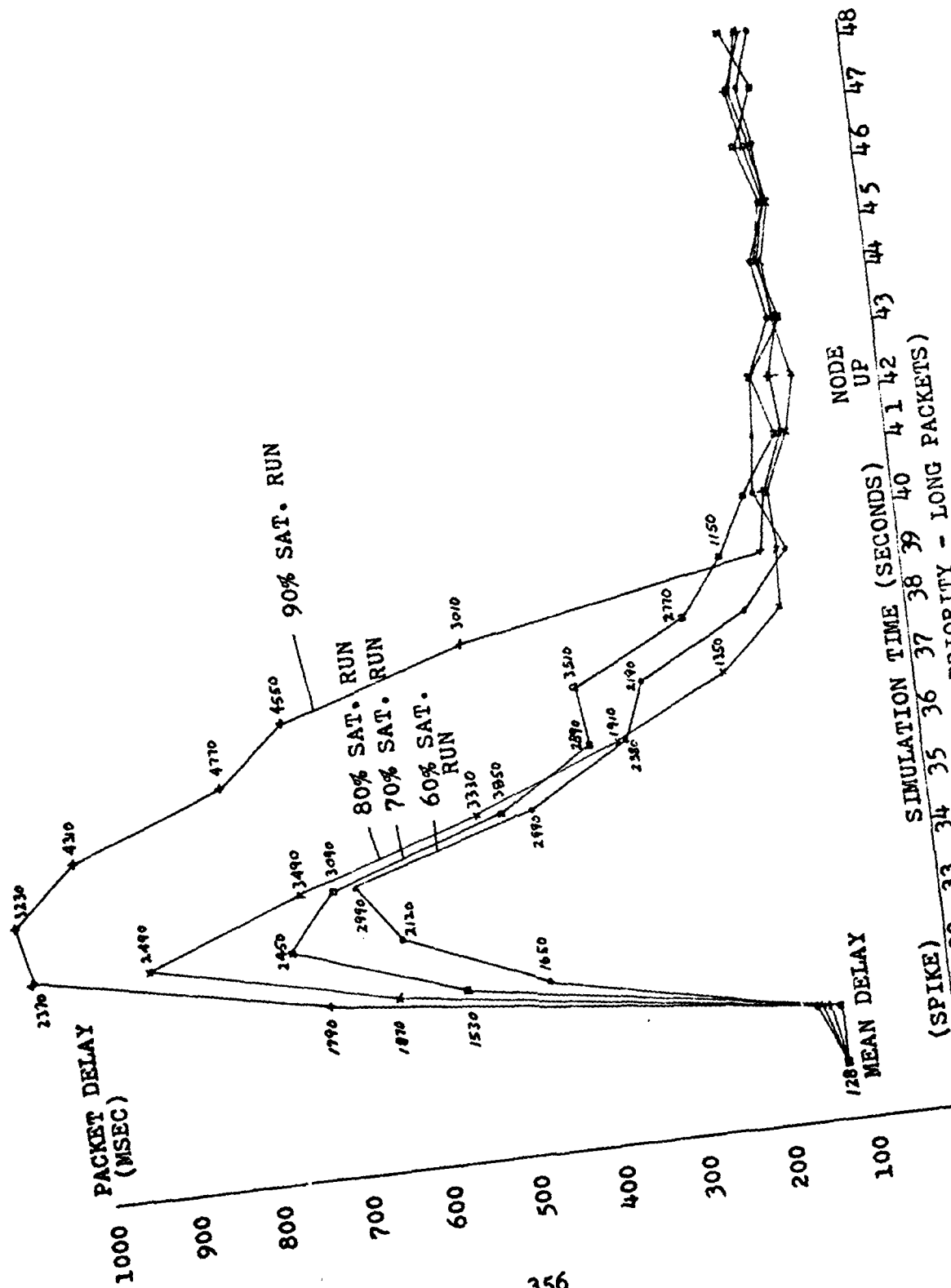


Fig 1-3.

TABLE I-I  
System Delay (priority - short packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.			
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max
30	69	230	69	230	3	230	69	230	69	230	69	230	69	230	69	230
31 (spike time)	106	150	3	3	3	3	131	210	131	210	93	190	93	190	93	190
32	-	-	33	33	-	-	-	-	-	-	52	52	52	52	52	52
33	87	130	-	-	-	-	-	-	-	-	3	3	3	3	3	3
34	124	170	-	-	-	-	44	44	44	44	3	3	3	3	3	3
35	-	-	3	3	3	3	171	171	171	171	80	80	80	80	80	80
36	35	35	-	-	-	-	-	-	-	-	-	-	-	-	-	-
37	-	-	-	-	-	-	3	3	3	3	-	-	-	-	-	-
38	-	-	-	-	-	-	7	7	7	7	-	-	-	-	-	-
39	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
40	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
41	-	-	3	3	3	3	33	33	33	33	-	-	-	-	-	-
42 (node up)	143	143	-	-	-	-	-	-	-	-	46	46	46	46	46	46
43	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
44	-	-	-	-	-	-	33	33	33	33	35	35	35	35	35	35
45	-	-	3	3	3	3	3	3	3	3	-	-	-	-	-	-
46	-	-	53	70	103	103	103	103	103	103	-	-	-	-	-	-
47	39	39	101	101	3	3	3	3	3	3	-	-	-	-	-	-
48	-	-	-	-	-	-	-	-	-	-	3	3	3	3	3	3

TABLE I-II  
System Delay (priority - long packets)

Simulation Time (seconds)	60% Sat.				70% Sat.				80% Sat.				90% Sat.	
	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max	Delay Times (milliseconds)	Run Max	Mean	Run Max
30	113	230	113	230	113	230	113	230	113	230	113	230	113	230
31 (spike time)	99	210	144	310	43	130	186	350	182	330	3	3	122	122
32	-	-	343	343	220	350	182	330	3	3	-	-	308	308
33	76	210	150	310	3	3	-	-	-	-	122	122	151	290
34	229	230	131	250	-	-	-	-	-	-	122	122	151	290
35	3	3	-	-	-	-	-	-	-	-	122	122	151	290
36	-	-	232	232	-	-	-	-	-	-	122	122	151	290
37	123	230	113	113	195	290	113	113	195	290	113	113	151	290
38	239	370	112	112	62	110	112	112	62	110	-	-	-	-
39	115	115	107	107	61	120	107	107	61	120	147	230	108	108
40	149	230	-	-	-	-	-	-	-	-	108	108	-	-
41	-	-	11	11	81	210	11	11	81	210	-	-	-	-
42 (node up)	78	130	-	-	-	-	-	-	-	-	-	-	-	-
43	-	-	184	184	56	110	184	184	56	110	3	3	-	-
44	59	110	58	110	3	3	58	110	3	3	-	-	-	-
45	106	106	55	110	209	209	55	110	209	209	209	209	209	209
46	72	110	110	110	363	363	110	110	363	363	-	-	-	-
47	-	-	-	-	186	186	-	-	186	186	-	-	-	-
48	224	224	-	-	-	-	-	-	-	-	3	3	-	-

Loss of Gentile

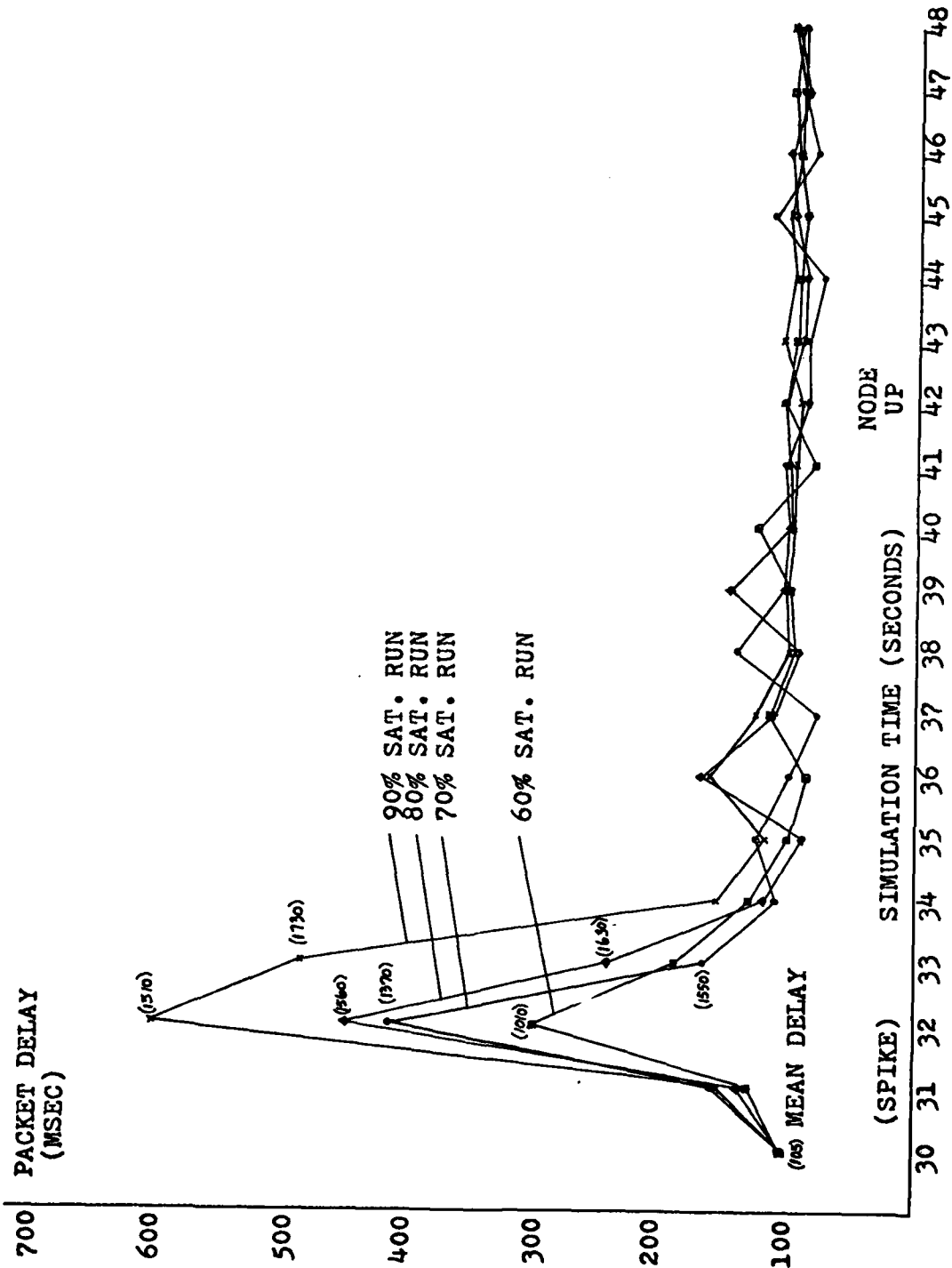


Fig I-4. SYSTEM DELAY (ALL PACKETS)

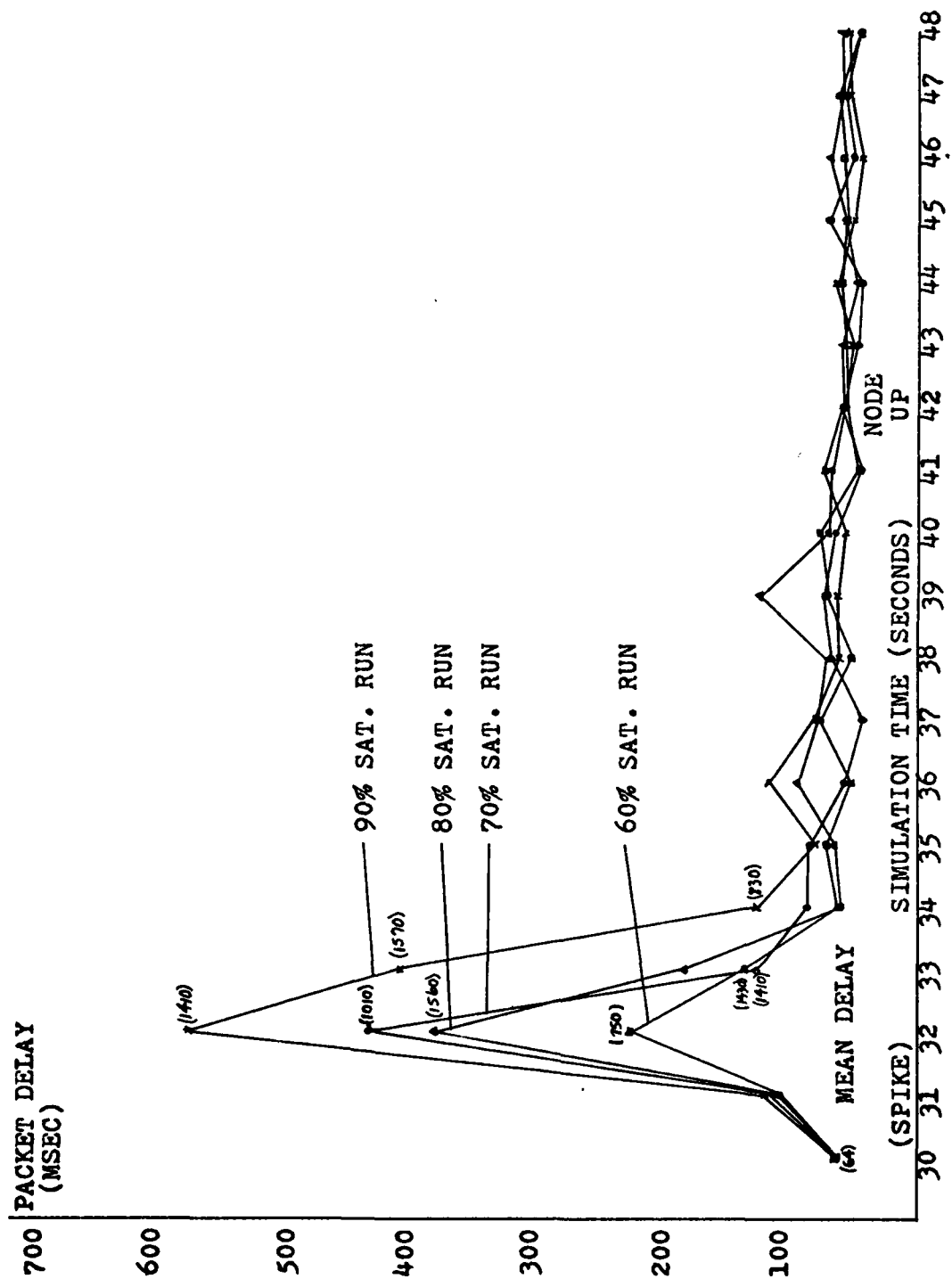


Fig I-5. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

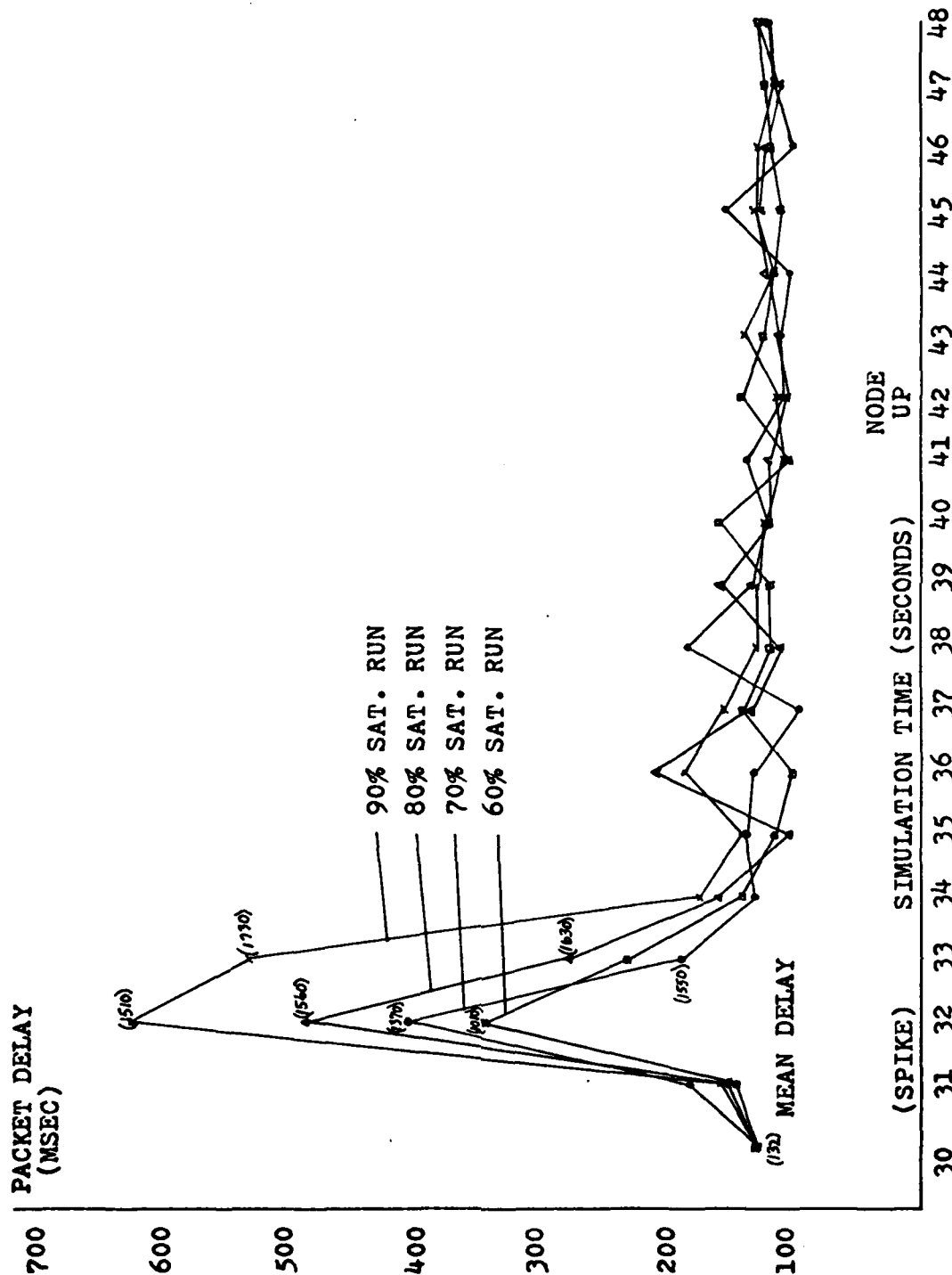


Fig I-6. SYSTEM DELAY (NON-PRIORITY - LONG PACKETS)

TABLE I-III  
System Delay (priority - short packets)

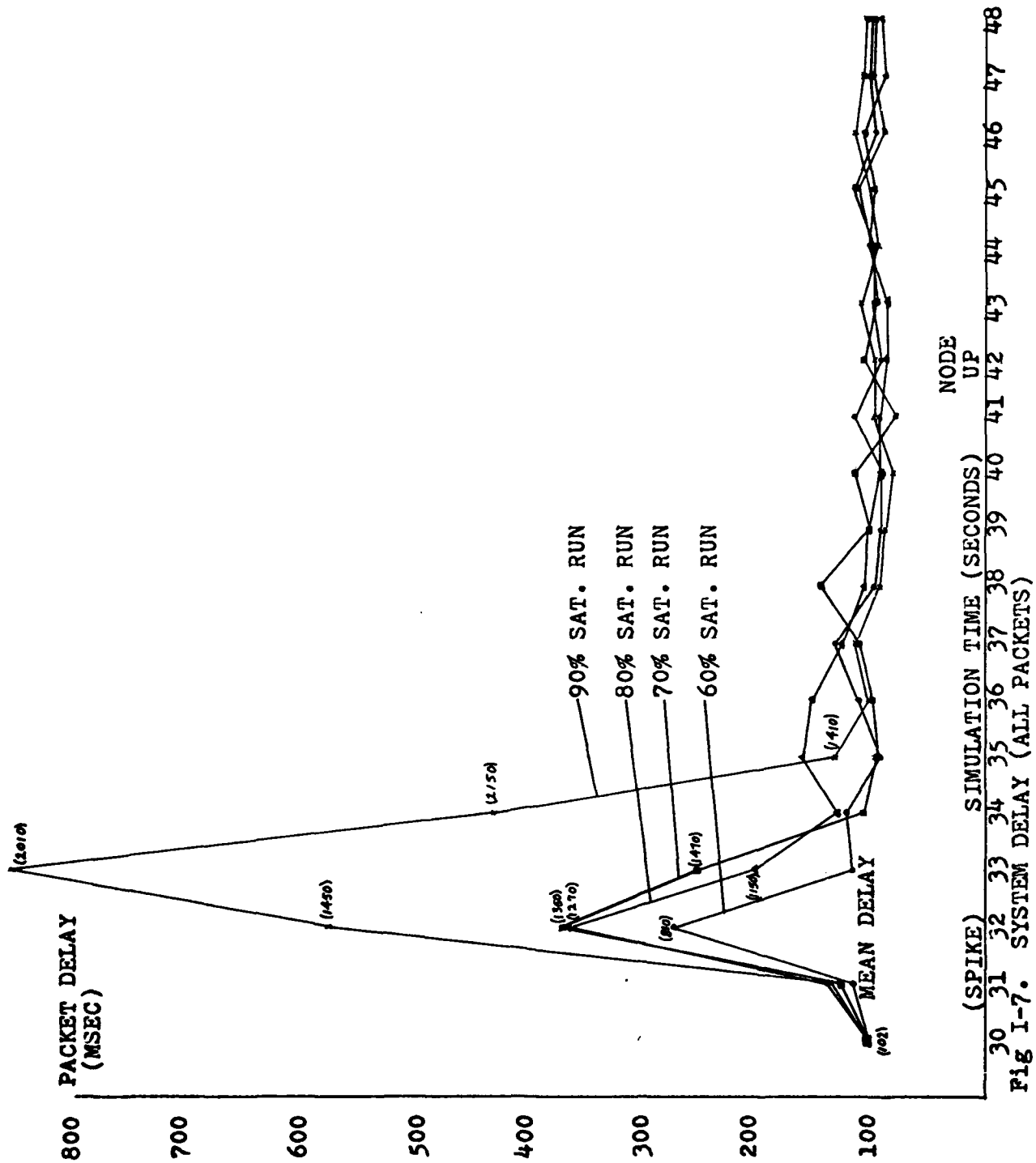
Simulation Time (seconds)	Delay Times (milliseconds)									
	60% Sat.		70% Sat.		80% Sat.		90% Sat.		Run	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
30	68	230	68	230	68	230	68	230	68	230
31 (spike time)	6	10	4	4	65	130	149	149	149	149
32	96	96	-	-	-	-	-	-	-	-
33	-	-	-	-	-	-	208	208	-	208
34	-	-	3	3	-	-	-	-	-	-
35	-	-	-	-	24	50	21	30	21	30
36	93	93	84	84	-	-	73	73	73	73
37	36	36	33	33	3	3	-	-	-	-
38	33	33	-	-	49	49	-	-	-	-
39	55	55	-	-	-	-	-	-	-	-
40	73	73	-	-	-	-	-	-	-	-
41	-	-	-	-	-	-	-	-	-	-
42 (node up)	-	-	-	-	112	112	-	-	-	-
43	-	-	-	-	-	-	-	-	-	-
44	-	-	-	-	-	-	-	-	-	-
45	-	-	-	-	67	67	39	39	39	39
46	-	-	-	-	49	70	-	-	-	-
47	33	33	39	39	-	-	3	3	3	3
48	-	-	-	-	-	-	-	-	-	-



TABLE I-IV  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)				90% Sat.			
	60% Sat. Mean	70% Sat. Mean	Run Max	80% Sat. Mean	Run Max	Mean	Run Max	Run Max
30	116	116	250	116	250	116	250	250
31 (spike time)	188	43	110	154	330	167	170	170
32	-	-	-	174	174	206	290	290
33	-	-	-	-	-	228	350	350
34	-	-	-	-	-	141	141	141
35	3	106	106	211	211	106	106	106
36	-	3	3	106	106	210	310	310
37	107	-	210	-	-	158	210	210
38	-	219	219	55	110	209	209	209
39	-	108	108	4	4	3	3	3
40	-	213	213	55	110	-	-	-
41	-	266	266	106	106	3	3	3
42 (node up)	-	-	-	-	-	-	-	-
43	-	188	250	-	-	-	-	-
44	-	3	3	-	-	57	110	110
45	3	37	110	3	3	151	190	190
46	184	73	310	257	257	-	-	-
47	-	199	199	110	110	223	223	223
48	-	-	-	-	-	109	110	110

Loss of Tinker



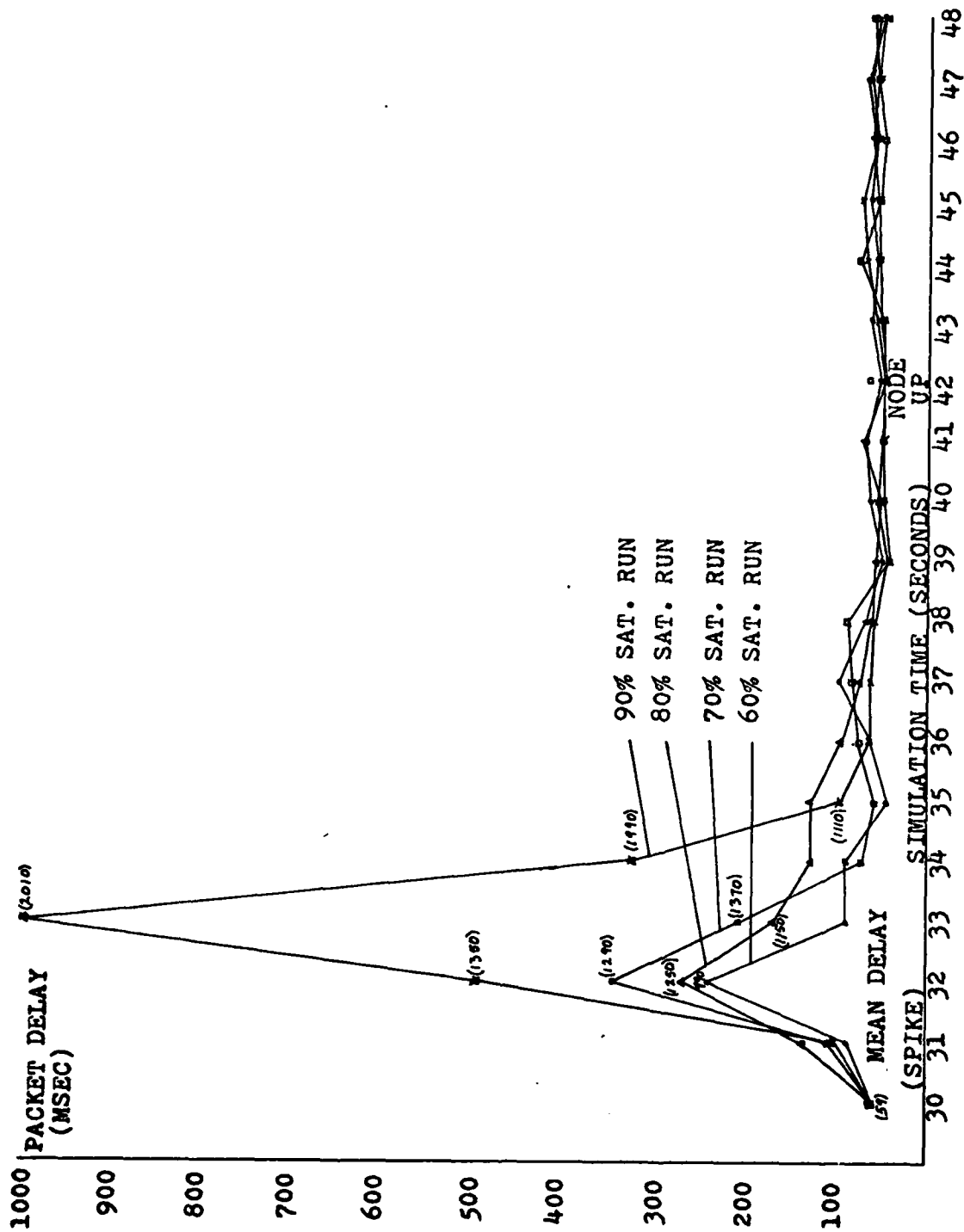


Fig I-8. SYSTEM DELAY (NON-PRIORITY - SHORT PACKETS)

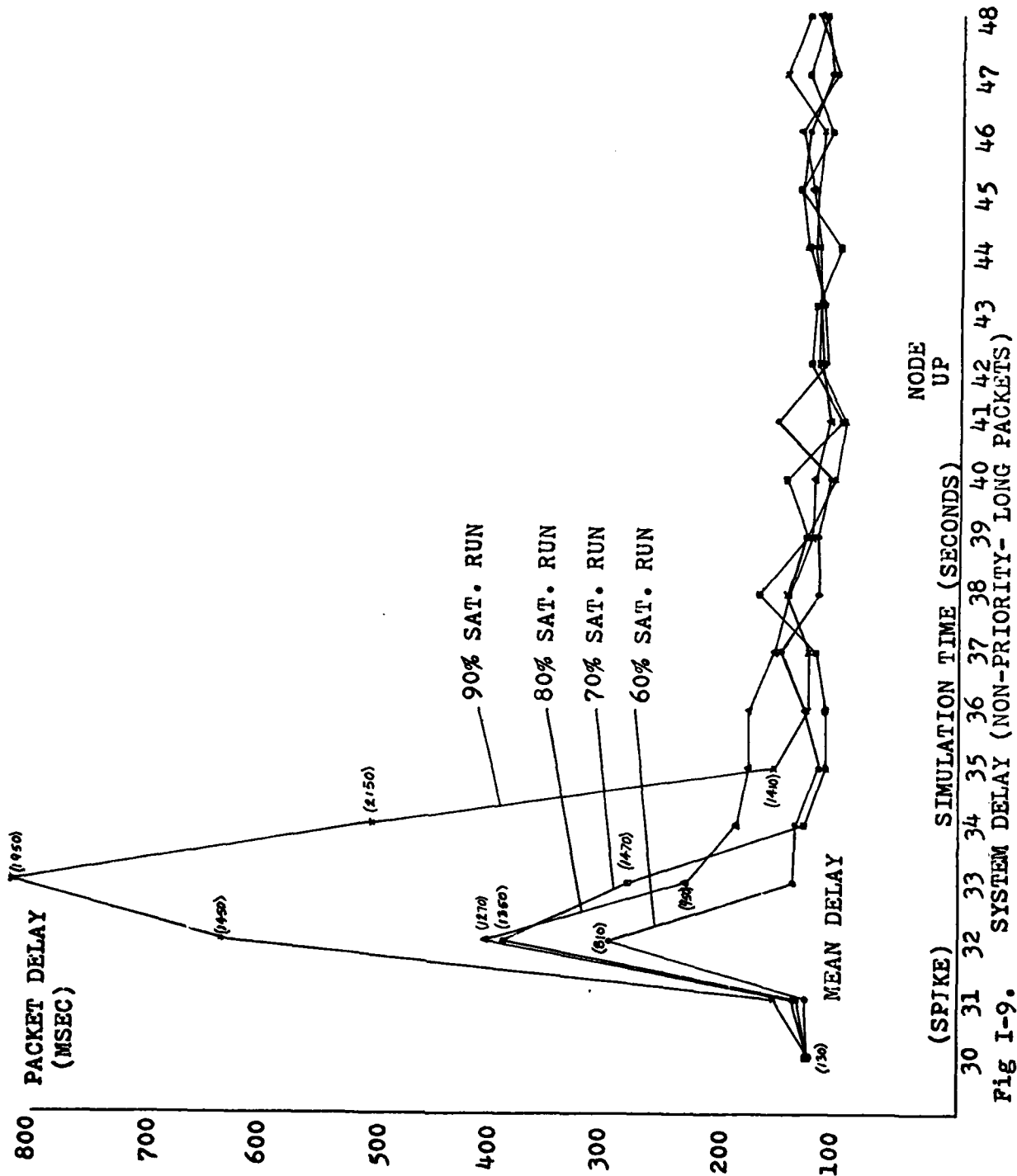


Fig I-9.

TABLE I-V  
System Delay (priority - short packets)

Simulation Time (seconds)	Delay Times (milliseconds)						90% Sat. Run	
	60% Sat. Mean	Run Max	70% Sat. Mean	Run Max	80% Sat. Mean	Run Max	Mean	Max
30	62	230	62	230	62	230	62	230
31 (spike time)	-	-	62	130	111	210	4	4
32	120	120	138	138	-	-	-	-
33	-	-	-	-	-	-	52	52
34	-	-	-	-	-	-	-	-
35	88	88	-	-	6	6	160	160
36	-	-	98	98	-	-	35	35
37	-	-	-	-	-	-	-	-
38	-	-	-	-	-	-	-	-
39	-	-	-	-	-	-	-	-
40	33	33	-	-	-	-	-	-
41	3	3	-	-	-	-	-	-
42 (node up)	-	-	-	-	-	-	-	-
43	33	33	-	-	-	-	-	-
44	143	143	-	-	152	152	-	-
45	-	-	-	-	43	43	74	74
46	-	-	119	119	3	3	-	-
47	80	130	-	-	-	-	-	-
48	33	33	6	6	-	-	-	-

TABLE I-VI  
System Delay (priority - long packets)

Simulation Time (seconds)	Delay Times (milliseconds)									
	60% Sat.		70% Sat.		80% Sat.		90% Sat.		Run	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max	Mean	Max
30	106	350	106	350	106	350	106	350	106	350
31 (spike time)	-	-	108	210	168	270	60	270	60	110
32	129	210	128	128	244	244	135	244	135	170
33	-	-	55	110	-	-	-	-	-	-
34	112	120	133	150	128	128	-	128	-	-
35	-	-	3	3	-	-	4	-	4	4
36	209	209	-	-	-	-	-	-	-	-
37	-	-	209	209	-	-	195	-	195	195
38	-	-	184	250	-	-	187	-	187	210
39	174	210	-	-	3	3	-	3	-	-
40	8	8	4	4	-	-	-	-	-	-
41	3	3	-	-	-	-	-	-	-	-
42 (node up)	106	106	132	132	5	5	-	5	-	-
43	-	-	-	-	-	-	108	-	108	110
44	-	-	109	210	-	-	95	-	95	190
45	106	106	113	210	148	230	-	230	-	-
46	134	150	161	210	3	3	114	3	114	114
47	-	-	106	106	-	-	-	-	-	-
48	8	8	-	-	-	-	-	-	-	-

### VITA

John A. Whittenton was born on 28 November 1948 in Fort Worth, Texas. He graduated from Spring Branch High School in Houston, Texas in 1967 and attended Baylor University from which he received the degree of Bachelor of Science in Physics in May 1972. After completing a short course in Communication - Electronics at Keesler Air Force Base, Mississippi, he was assigned to the First Aerospace Communications Group Command at Offutt Air Force Base, Nebraska in June 1973, as an Automated Systems Program Designer. In May 1976, he was reassigned as an Automated Systems Analyst to the AUTODIN I communications computer switch at Andrews Air Force Base, Maryland, until entering the School of Engineering, Air Force Institute of Technology, in July 1978.

Permanent Address: 2328 Hoskins Drive  
Houston, Texas 77080



AD-A080 484 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCH00--ETC P/6 17/2  
AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II.(U)  
DEC 79 J A WHITTENTON  
UNCLASSIFIED AFIT/SCS/EE/79-15 NL

5-5

SCS

AD-A080 484



END

DATE

TIME

3 - 80

FOR

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/79-15	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) AN ANALYSIS OF A ROUTING ALGORITHM FOR AUTODIN II		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John A. Whittenton Capt USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Communications Engineering Center DCEC Reston VA		12. REPORT DATE December 1979
		13. NUMBER OF PAGES 371
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Defense Communications Engineering Center (DCEC) Reston, Virginia		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES Approved for public release; IAW AFR 190-17 J.P. Hipps, Major, USAF Director of Public Affairs		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Computer Communications Network AUTODIN II Routing Algorithm Packet-Switching Center		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A general overview of the AUTODIN II computer communications network is given. The proposed hierarchical routing algorithm is discussed with detailed explanation of each of the six modules. The GPSS simulation language is discussed with particular emphasis on how it was used in the simulation program to model the AUTODIN II network. Examples of various computer simulation runs are given with detail on how the program was changed to simulate the various traffic volume changes and network topology changes. An analysis section discusses		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

next  
page

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

packet delay and defines two parameter which are used to determine the adaptability of the routing algorithm. The report concludes with conclusions on the algorithm's ability to adapt to traffic volume fluctuations and network topology changes. Recommendations on further simulation exercises and possible changes to network connectivity and to the routing algorithm itself are also given in the last section.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)